Anytime Prediction: Efficient Ensemble Methods for Any Computational Budget

Alexander Grubb

January 21, 2014
CMU-CS-14-100

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
J. Andrew Bagnell, Chair
Avrim Blum
Martial Hebert
Alexander Smola
Hal Daumé III, University of Maryland

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

This work supported by ONR MURI grant N00014-09-1-1052 and the U.S Army Research Laboratory under the Collaborative Technology Alliance Program, Cooperative Agreement W911NF-10-2-0016

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
21 JAN 2014		**00-00-2014 to 00-00-2014**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Anytime Prediction: Efficient Ensemble Methods for Any Computational Budget	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,School of Computer Science,Computer Science Department,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

A modern practitioner of machine learning must often consider trade-offs between accuracy and complexity when selecting from available machine learning algorithms. Prediction tasks can range from requiring real-time performance to being largely unconstrained in their use of computational resources. In each setting, an ideal algorithm utilizes as much of the available computation as possible to provide the most accurate result. This issue is further complicated by applications where the computational constraints are not fixed in advance. In many applications predictions are often needed in time to allow for adaptive behaviors which respond to real-time events. Such constraints often rely on a number of factors at prediction time, making it difficult to select a fixed prediction algorithm a priori. In these situations an ideal approach is to use an anytime prediction algorithm. Such an algorithm rapidly produces an initial prediction and then continues to refine the result as time allows, producing final results which dynamically improve to fit any computational budget. Our approach uses a greedy, cost-aware extension of boosting which fuses the disparate areas of functional gradient descent and greedy sparse approximation algorithms. By using a cost-greedy selection procedure our algorithms provide an intuitive and effective way to trade-off computational cost and accuracy for any computational budget. This approach learns a sequence of predictors to apply as time progresses, using each new result to update and improve the current prediction as time allows. Furthermore, we present theoretical work in the different areas we have brought together, and show that our anytime approach is guaranteed to achieve near-optimal performance with respect to unknown prediction time budgets. We also present the results of applying our algorithms to a number of problem domains such as classification and object detection that indicate that our approach to anytime prediction is more efficient than trying to adapt a number of existing methods to the anytime prediction problem. We also present a number of contributions in areas related to our primary focus. In the functional gradient descent domain, we present convergence results for smooth objectives, and show that for non-smooth objectives the widely used approach fails both in theory and in practice. To rectify this we present new algorithms and corresponding convergence results for this domain. We also present novel, time-based versions of a number of greedy feature selection algorithms and give corresponding approximation guarantees for the performance of these algorithms.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**163**	

Keywords: anytime prediction, budgeted prediction, functional gradient methods, boosting, greedy optimization, feature selection

Abstract

A modern practitioner of machine learning must often consider trade-offs between accuracy and complexity when selecting from available machine learning algorithms. Prediction tasks can range from requiring real-time performance to being largely unconstrained in their use of computational resources. In each setting, an ideal algorithm utilizes as much of the available computation as possible to provide the most accurate result.

This issue is further complicated by applications where the computational constraints are not fixed in advance. In many applications predictions are often needed in time to allow for adaptive behaviors which respond to real-time events. Such constraints often rely on a number of factors at prediction time, making it difficult to select a fixed prediction algorithm a priori. In these situations, an ideal approach is to use an anytime prediction algorithm. Such an algorithm rapidly produces an initial prediction and then continues to refine the result as time allows, producing final results which dynamically improve to fit any computational budget.

Our approach uses a greedy, cost-aware extension of boosting which fuses the disparate areas of functional gradient descent and greedy sparse approximation algorithms. By using a cost-greedy selection procedure our algorithms provide an intuitive and effective way to trade-off computational cost and accuracy for any computational budget. This approach learns a sequence of predictors to apply as time progresses, using each new result to update and improve the current prediction as time allows. Furthermore, we present theoretical work in the different areas we have brought together, and show that our anytime approach is guaranteed to achieve near-optimal performance with respect to unknown prediction time budgets. We also present the results of applying our algorithms to a number of problem domains such as classification and object detection that indicate that our approach to anytime prediction is more efficient than trying to adapt a number of existing methods to the anytime prediction problem.

We also present a number of contributions in areas related to our primary focus. In the functional gradient descent domain, we present convergence results for smooth objectives, and show that for non-smooth objectives the widely used approach fails both in theory and in practice. To rectify this we present new algorithms and corresponding convergence results for this domain. We also present novel, time-based versions of a number of greedy feature selection algorithms and give corresponding approximation guarantees for the performance of these algorithms.

For Max.

Acknowledgments

This document would certainly not exist without the support and input of my advisor, Drew Bagnell. His willingness to pursue my own ideas and his ability to guide me when my own insight failed have been invaluable. I'm forever indebted to him for the immense amount of things I have learned from him over the years. My only hope for my research career is that I can approach future problems with the same eye for deep, insightful questions that Drew has brought to our work.

I would also like to thank the rest of my committee: Avrim Blum, Martial Hebert, Alex Smola, and Hal Daumé III. Their time and input has improved this work in so many ways. Though not on this list, Geoff Gordon has also provided many helpful insights and discussions and deserves an unofficial spot here.

I have been fortunate enough to have a number of other mentors and advisors that have helped me reach this point. I'm very grateful to those who helped introduce me to academic research and advised me throughout my undergraduate and early graduate career, particularly Dave Touretzky and Paul Rybski.

I'd also like to thank Steven Rudich. The Andrew's Leap program that he runs for Pittsburgh-area high school students fostered my interest in Computer Science very early on and ultimately led me to where I am today. Were it not for my involvement with this program so many years ago, I can't imagine how my career might have unfolded.

Throughout this work I've had the opportunity to collaborate with a number of great people: Elliot Cuzzillo, Felix Duvallet, Martial Hebert, Hanzhang Hu, and Dan Muñoz. Thank you for sharing your work with me and giving me the opportunity to share mine. I am also grateful to Dave Bradley and Nathan Ratliff. Their work on functional gradient methods helped greatly in shaping and focusing my early work. This work would also not have been possible without a number of informal collaborations. The conversations and insights gleaned from Debadeepta Dey, Stéphane Ross, Suvrit Sra, Kevin Waugh, Brian Ziebart and Jiaji Zhou, as well as the entire LairLab, have been immensely helpful.

The community and atmosphere at Carnegie Mellon, especially the Computer Science program have been wonderful throughout this process. My love of Pittsburgh is no secret, but the journey

that is graduate school is notorious for being difficult wherever you are. That my own personal journey was as enjoyable as it was is a testament to the great people here. To name a few: Jim Cipar, John Dickerson, Mike Dinitz, Jason Franklin, Sam Ganzfried, Tony Gitter, Severin Hacker, Elie Krevat, Dan Muñoz, Abe Othman, Stéphane Ross, Jiří Šimša, Jenn Tam, Kevin Waugh, and Erik Zawadzki. Your friendship has been greatly uplifting.

Finally, I am most grateful to my family. I thank my parents, Suzan and Bob, for the unending support and opportunity that they have provided throughout my life. This work is as much a product of their effort as it is mine. As for my wife, Sarah, there is no amount of gratitude I can give that can compensate for the amount of love, support and patience she has shown me. Thank you.

Contents

Chapter 1

Introduction

When analyzing the performance of any machine learning approach, there are often two critical factors considered: the predictive accuracy of the algorithm and the cost or strain on resources of using a given algorithm. Furthermore, these two metrics of accuracy and cost are typically opposed to each other. Increasing the accuracy of an algorithm often requires increasing the complexity of the underlying model, which comes with an increase in cost, and vice versa. This trade-off between cost and accuracy is an inherently difficult problem and is the focus of this work.

1.1 Motivation

The number of machine learning applications which involve real time and latency sensitive predictions is growing rapidly. In areas such as robotics, decisions must be made on the fly and in time to allow for adaptive behaviors which respond to real-time events. In computer vision, prediction algorithms must often keep up with high resolution streams of live video from multiple sources without sacrificing accuracy. Finally, prediction tasks in web applications must be carried out with response to incoming data or user input without significantly increasing latency, and the computational costs associated with hosting a service are often critical to its viability. For such applications, the decision to use a larger, more complex predictor with higher accuracy or a less accurate, but significantly faster predictor can be difficult.

To this end, we will focus on the prediction or test-time cost of a model in this work, and the problem of trading-off between prediction cost and accuracy. While the cost of building a model is an important consideration, the advent of cloud computing and a large increase in the general computing power available means that the resources available at training time are often much less constrained than the prediction time requirements. When balancing training costs, concerns such as scalability and tractability are often more important, as opposed to factors such as latency which are more directly related to the complexity of the model.

The problem of trading-off prediction cost and accuracy is considered throughout the literature, both explicitly and implicitly. Implicitly, reduced model complexity, in the form of reduced

memory or computational requirements, is a feature often used to justify reduced accuracy when comparing to previous work. In other settings, entirely new algorithms are developed when models are too costly for a given application.

Explicitly, there are is a wide array of approaches for generating models of various complexity and comparing their predictive performance. We will now discuss some of the existing approaches to this problem.

Varying Model Complexity Directly

In many settings, model complexity can often be tuned directly. In tuning these parameters and comparing performance, the cost and accuracy trade-off is presented directly, and the practicioner is able to choose from among these points directly in an ad-hoc way, typically selecting the highest accuracy model which fits within their computational constraints. This approach is closest to the implicit approach of discussing cost and accuracy trade-offs, where the trade-off is considered external to the learning problem itself.

Examples include:

- Tuning the number and structure of hidden units in a neural network.
- Tuning the number of exemplars used in an exemplar-based method.
- Tuning the number of weak learners used in an ensemble method.
- Manually selecting different sets of features or feature transformations to train the model on.

Constraining the Model

Related to the previous approach, another way to trade-off cost and accuracy is to specify some kind of constraint on the cost of the model. In constrast to the previous approach, the constraint is usually made explicit at training time, and the learning algorithm optimizes the accuracy directly with knowledge of the constraint.

Under this regime, the constraint can be related to the complexity of the model, but is often more directly related to the prediction cost of interest. Since these constraints are not directly related to the complexity of the model, they often require new algorithms and methods for optimizing the model subject to such a constraint.

Examples of this method include:

- Assuming that each feature used by a model has some computational cost and using some kind of budgeted feature selection approach. In this setting, a model can have a high complexity and still have low cost, as the feature computation time is the dominant factor.
- Using a dimensionality reduction or sparse coding technique to otherwise reduce the dimensionality of inputs and eventual computational cost.

Regularizing the Model

Under this approach, one augments the accuracy objective being optimized with a cost term, and then optimizes the model to fit this single combined objective. By adjusting the importance of the cost and accuracy factors in the objective, the model will select some specific point on the spectrum of possible trade-offs.

While regularization is often used to reduce the complexity of a model to *improve* accuracy, *e.g.* eliminating error due to overfitting, it can also be used to reduce the complexity of the model to handle a scarcity of prediction time resources.

Examples include:

- Using a regularized version of the constraint used in the budgeted feature selection problem and optimizing the model using in this constraint.
- Using a heavily regularized objective to increase the sparsity of a model and hence the processor and memory usage of that model.

Generating Approximate Predictions

One final approach that is substantially different from the previous ones is to use some fixed, potentially expensive, model, but improve the cost of obtaining predictions from that model directly. This can be achieved by computing approximations of the predictions the model would generate if given increased computational resources. This can either be done by generating an approximate version of the model for use in prediction, or using some faster but less accurate algorithm for prediction.

Examples include:

- Using an approximate inference technique in a graphical model.
- Searching only a portion of the available space in a search-based or exemplar based method.
- Using techniques such as cascades or early-exits to improve the computation time of ensemble predictions.

Almost all the techniques described above are best utilized when the computation constraints are well understood at training time, however. With the exception of a few algorithms which fall in to the approximate prediction category, each method requires the practitioner to make descisions at training time which will affect the resulting trade-off of cost and accuracy made by the model. This is a significant drawback, as it requires the practitioner to understand the cost accuracy trade-off at some level and understand it a priori.

In practice, making this trade-off at training time can have adverse effects on the accuracy of future predictions. In many settings, such as cloud computing, the available computational resources may change significantly over time. For instance, prices for provisioning machines may

vary significantly depending on the time of day, or idle machines may be able to be utilized in an on demand manner to improve predictions.

In other settings, the resources may change due to the nature of the problem or environment. For example, an autonomous agent may want to use a reasonably fast method for predicting object locations as a first attempt when performing some task, but would like a slower, more accurate method of last resort should the first attempt fail.

As a final example, consider the problem of generating batch predictions on a large set of test examples. For example, in the object detection domain a large number of examples are generated from a single input image, corresponding to each of the locations in the image. Some of these examples, such as cluttered areas of the image, may be inherently more difficult to classify than other examples, such as a patch of open sky. In this setting our computational constraint is actually on the prediction cost of the batch of examples *as a whole* and not on each single example. To obtain high accuracy at low cost, any prediction method would ideally focus its efforts on the more difficult examples in the batch. Using any one of the fixed methods above, we would spend the same amount of time on each example in the batch, resulting in less efficient use of resources.

1.2 Approach

To handle many of the failure situations described above, it would be useful to work with prediction algorithms capable of initially giving crude but rapid estimates and then refining the results as time allows. For situations where the computational resources are not known apriori, or where we would like to dynamically adapt the resources used, such an algorithm can automatically adjust to fill any allocated budget at test-time.

For example, in a robotics application such as autonomous navigation, it may sometimes be the case that the robot can rapidly respond with predictions about nearby obstacles, but can spend more time reasoning about distant ones to generate more accurate predictions.

As first studied by Zilberstein [1996], *anytime algorithms* exhibit exactly this property of providing increasingly better results given more computation time. Through this previous work, Zilberstein has identified a number of desirable properties for an anytime algorithm to possess. In terms of predictions these properties are:

- Interruptability: a prediction can be generated at any time.
- Monotonicity: prediction quality is non-decreasing over time.
- Diminishing Returns: prediction quality improves fastest at early stages.

An algorithm meeting these specifications will be able to dynamically adjust its predictions to fit within any test-time budget, avoiding the need to make reason about computational constraints at training time.

> **Thesis Statement:** As it is often difficult to make decisions which trade off final cost and accuracy a priori, we should instead seek to train predictors which dynamically adjust the computations performed at prediction time to meet any computational budget. The algorithms for generating these predictors should further be able to reason about the cost and benefit of each element of a prediction computation to automatically select those which improve accuracy most efficiently, and should do so without knowing the test-time budget apriori.

Our work targets these specific properties using a hybrid approach. To obtain the incremental, interruptable behavior we would like for updating predictions over time we will learn an additive ensemble of weaker predictors. This work will build off the previous work in this area of boosted ensemble learning [Schapire, 2002], specifically the functional gradient descent approach [Mason et al., 1999, Friedman, 2000] for generalizing the behavior of boosting to arbitrary objectives and weak predictors. In these approaches we learn a predictor which is simply the linear combination of a sequence of weak predictors. This final predictor can easily be made interruptable by evaluating the weak predictors in sequence and computing the linear combination of the outputs whenever a prediction is desired.

We will augment the standard functional gradient approach with ideas taken from greedy selection algorithms [Tropp, 2004, Streeter and Golovin, 2008, Das and Kempe, 2011] typically used in the submodular optimization and sparse approximation domains. We will use a cost-greedy version of functional gradient methods which select the next weak predictor based on an improvement in accuracy scaled by the cost of the weak learner. This cost-greedy approach ensures that the we select sequences of weak predictors that increase accuracy as efficiently as possible, satisfying the last two properties.

As we will show later, this relatively simple framework can be applied to a wide range of problems. Furthermore, we provide theoretical results that show that this method is guaranteed to achieve near-optimal performance as the budget increases, without knowing the specific budget apriori, and observe that this near-optimality holds in a number of experimental applications.

1.3 Related Work

We now detail a number of approaches and previous work that are related both to the focus of this work, and the disparate areas we fuse together in our methods and analysis.

Boosting and Functional Gradient Methods

Boosting is a versatile meta-algorithm for combining together multiple simple hypotheses, or weak predictors, to form a single complex hypothesis with superior performance. The power of this meta-algorithm lies in its ability to craft hypotheses which can achieve arbitrary performance on

training data using only weak learners that perform marginally better than random. Schapire [2002] give a very good overview of general boosting techniques and applications.

To date, much of the work on boosting has focused on optimizing the performance of this meta-algorithm with respect to specific loss functions and problem settings. The AdaBoost algorithm [Freund and Schapire, 1997] is perhaps the most well known and most successful of these. AdaBoost focuses specifically on the task of classification via the minimization of the exponential loss by boosting weak binary classifiers together, and can be shown to be near optimal in this setting. Looking to extend upon the success of AdaBoost, related algorithms have been developed for other domains, such as RankBoost [Freund et al., 2003] and mutliclass extensions to AdaBoost [Mukherjee and Schapire, 2010]. Each of these algorithms provides both strong theoretical and experimental results for their specific domain, including corresponding weak to strong learning guarantees, but extending boosting to these and other new settings is non-trivial.

Recent attempts have been successful at generalizing the boosting approach to certain broader classes of problems, but their focus is also relatively restricted. Mukherjee and Schapire [2010] present a general theory of boosting for multiclass classification problems, but their analysis is restricted to the multiclass setting. Zheng et al. [2007] give a boosting method which utilizes the second-order Taylor approximation of the objective to optimize smooth, convex losses. Unfortunately, the corresponding convergence result for their algorithm does not exhibit the typical weak to strong guarantee seen in boosting analyses and their results apply only to weak learners which solve the weighted squared regression problem.

Other previous work on providing general algorithms for boosting has shown that an intuitive link between algorithms like AdaBoost and gradient descent exists [Mason et al., 1999, Friedman, 2000], and that many existing boosting algorithms can be reformulated to fit within this gradient boosting framework. Under this view, boosting algorithms are seen as performing a modified gradient descent through the space of all hypotheses, where the gradient is calculated and then used to find the weak hypothesis which will provide the best descent direction.

In the case of smooth convex functionals, Mason et al. [1999] give a proof of eventual convergence for the functional gradient method. This result is similar to the classical convergence result given in Zoutendijk's Theorem [Zoutendijk, 1970], which gaurantees convergence for a variety of descent-based optimization algorithms, as long as the search direction at every iteration is sufficiently close to the gradient of the function. Additionally, convergence rates of these algorithms have been analyzed for the case of smooth convex functionals [Rätsch et al., 2002] and for specific potential functions used in classification [Duffy and Helmbold, 2000] under the traditional PAC weak learning setting. Our result [Grubb and Bagnell, 2011] extends these results for smooth losses, and also introduces new results and algorithms for non-smooth losses.

Submodular Maximization

In our analysis of the cost-greedy approaches in this document, we will make heavy use of the submodular set function maximization framework. A good overview of this domain is given in the

survey of submodular function maximization work by Krause and Golovin [2012].

Most relevant to our work are the approaches for the budgeted or knapsack constrained submodular maximization problem. In this setting each element is assigned a cost and the constraint is on the sum of costs of elements in the selected set, similar to the knapsack problem [Mathews, 1897], which is the modular complement to this setting.

In this domain the original greedy algorithm for submodular function maximization with a cardinality constraint [Nemhauser et al., 1978] can be extended using a cost-greedy approach [Khuller et al., 1999].

A number of results from previous work [Khuller et al., 1999, Krause and Guestrin, 2005, Leskovec et al., 2007, Lin and Bilmes, 2010] have given variations on the cost-greedy algorithm which do have approximation bounds with factors of $\frac{1}{2}(1-\frac{1}{e})$ and $(1-\frac{1}{e})$. Unfortunately, these algorithms all require apriori knowledge of the budget in order to achieve the approximation bounds, and cannot generate single, budget agnostic sequences with approximation bounds for all budgets.

Unfortunately as Khuller et al. [1999] show, the standard approximation results do not hold directly for the cost-greedy algorithm. As we show in Chapter 4, in general there is no single budget agnostic algorithm which can achieve a similar approximation guarantee, but we can achieve approximation guarantees for certain budgets. Most directly related to out work, we will build off of the cost-greedy analysis of Streeter and Golovin [2008], which gives an approximation guarantee for certain budgets dependent on the problem.

Finally, our work will build off of previous work analyzing functions that are approximately submodular. Das and Kempe [2011] give a multiplicative version of approximate submodularity called the *submodularity ratio*. Similarly, Krause and Cehver [2010] give a version of submodularity that includes an additive error term. This additive error term is similar to the additive error terms utilized in analyzing online submodular maximization approaches [Krause and Guestrin, 2005, Streeter and Golovin, 2008, Ross et al., 2013]. Our work later in this document combines both additive and multiplicative relaxations of the standard submodularity definition.

Sparse Approximation

A common framework for controlling the complexity of a model is the sparse approximation problem, also referred to as subset selection, sparse decomposition, and feature selection. In this setting we are given a target signal or vector, and a set of basis vectors to use to reconstruct the target. These basis vectors are often referred to as atoms, bases, dictionary elements, and, when taken as a whole, a dictionary or design matrix. The goal is to select a sparse set of these vectors that best approximates the target, subject to some sparsity constraint. An equivalent formuation is to select a sparse weight vector with which to combine the basis vectors.

We will later use the sparse approximation framework to analyze our anytime approach. In general it can be shown that this problem is NP hard Natarajan [1995], but a number of practical approaches with approximation and regret bounds have been developed in previous work.

Most relevant to this work are the works on analyzing greedy feature selection algorithms

[Krause and Cehver, 2010, Das and Kempe, 2011] which build off of submodular maximization techniques. Krause and Cehver [2010] give an analysis of the dictionary selection problem, a variant of the subset selection problem where the goal is to select a larger subset or dictionary of which smaller subsets can be used to approximate a large set of different targets. Their analysis relies on the incoherency of the dictionary elements, a geometric property which captures the non-orthogonality of the dictionary. Das and Kempe [2011] give a similar analysis for the subset selection and dictionary selection problems, but use spectral properties of the dictionary elements which also captures the degree of orthogonality.

Greedy approaches to solving this problem include Forward-Stepwise Regression or simply Forward Regression [Miller, 2002], Matching Pursuit [Mallat and Zhang, 1993] also known as Forward-Stagewise Regression, and Orthogonal Matching Pursuit [Pati et al., 1993]. Forward Regression greedily selects elements which maximially improve reconstruction error when added to the set of bases, while the matching pursuit approaches select elements based on their correlation with the residual error remaining in the target at each iteration. Tropp [2004] gives a good analysis of the Orthogonal Matching Pursuit algorithm which uses the same incoherency parameter used by Krause and Cehver [2010], to show near-optimal reconstruction of the target.

Another popular approach to the sparse approximation problem is to use a convex relaxation of the sparsity constraint as a regularizer, and optimize the regularized objective directly. Examples include the Lasso algorithm [Tibshirani, 1996], Basis Pursuit [Chen et al., 2001], and Least-Angle Regression [Efron et al., 2004]. All of these algorithms optimize the L1 relaxation of the sparsity constraint using different methods.

For the L1-based regularization approaches, there are two main focuses for proving the algorithms are successful. One [Geer and Buhlmann, 2009] shows that the near-orthogonality of the vectors being selected implies that the proper subset is selected with high probability. This analysis relies on the RIP, or Restricted Isometry Property. The other [Juditsky and Nemirovski, 2000] approach derives regret bounds with respect to a sparse, L1 bounded linear combination of the variables, and shows that magnitude of sparse vector used for combination is the key factor for the bound.

We will follow a similar tack as the weight-based analysis of Juditsky and Nemirovski [2000] in our work here. The existing bounds discussed above for greedy sparse approximation approaches all use geometric properties, similar to the RIP property. In our work we would instead like to focus on bounds derived using other properties which depend on the magnitude of the combining weights being small, and not the underlying features being nearly orthogonal.

One final piece of the related literature that is related to our area of study is the work that has been done on the simultaneous sparse approximation problem. This problem is similar to the dictionary selection problem of Krause and Cehver [2010], in that we want to select a subset of bases that reconstruct a set of multiple target vectors well. The key difference between these two problems is that dictionary selection allows for the selection of a larger set of elements than is used to reconstruct any one target, while the simultaneous sparse approximation problem uses the same subset for every single target.

There exist both greedy and regularization approaches to solving this problem. Simultaneous Orthogonal Matching Pursuit [Cotter et al., 2005, Chen and Huo, 2006, Tropp et al., 2006] is a greedy method for solving this problem, based on the single target OMP approach. In the regularization or relaxation approaches, the corresponding relaxation of the sparsity constraint uses an L_p - L_q mixed norm, typically an L_1 norm of another, non-sparsity inducing norm, such as the L_2 or L_∞ norm. The approaches for solving this problem are called Group Lasso [Meier et al., 2008, Rakotomamonjy, 2011] algorithms, and select weight matrices that are sparse across features or basis vectors, but dense across the target vectors, giving the desired sparse set of selected bases.

Budgeted Prediction

Our primary focus in this work is on the trade-off between prediction cost and accuracy. Particularly for functional gradient methods and related ensemble approaches, there have been a number of previous approaches that attempt to tackle the prediction cost and accuracy trade-off.

This focus in the budgeted prediction setting, also called budgeted learning, test-time cost-sensitive learning, and resource efficient machine learning, is to try and automatically make this trade-off in ways that improve the cost of achieving good predictions.

Similar to our work, a number of approaches have considered methods for improving the prediction costs of functional gradient methods. Chen et al. [2012] and Xu et al. [2012] give regularization based methods for augmenting the functional gradient approach to account for cost. Their focus is on optimizing the feature computation time of a model, and attempts to select weak learners which use cheap or already computed features. The first approach [Chen et al., 2012] does this by optimizing the ordering and composition of a boosted ensemble after learning using a traditional functional gradient approach. The second method directly augments the weak learner training procedure (specifically, regression trees) with a cost-aware regularizer. This work has also been extended to include a variant which uses a branching, tree-based structure [Xu et al., 2013b], and a variant suitable for anytime prediction [Xu et al., 2013a] following the interest in this domain. This latter work uses a network of functional gradient modules, and backpropagates a functional gradient through the network, in a manner similar to Grubb and Bagnell [2010].

Another approach for augmenting the functional gradient approach is the sampling-based approach of Reyzin [2011], which uses randomized sampling at prediction time weighted by cost to select which weak hypotheses to evaluate.

An early canonical approach for improving the prediction time performance of these additive functional models is to use a *cascade* [Viola and Jones, 2001, 2004]. A cascade uses a sequence of increasingly complex classifiers to sequentially select and eliminate examples for which the predictor has high confidence in the current prediction, and then continues improving predictions on the low confidence examples. The original formulation focuses on eliminating negative examples, for settings where positive examples are very rare such as face detection, but extensions that eliminate both classes [Sochman and Matas, 2005] exist.

Many other variations on building and optimizing cascades exist [Saberian and Vasconcelos,

2010, Brubaker et al., 2008], and Cambazoglu et al. [2010] give a version of functional gradient methods which use an early-exit strategy, similar to the cascade approach, which generates predictions early if the model is confident enough in the current prediction. All these methods typically target final performance of the learned predictor, however. Furthermore, due to the decision making structure of these cascades and the permanent nature of prediction decisions, these models must be very conservative in making early decisions and are unable to recover from early errors. All of these factors combine to make cascades poor anytime predictors.

Another, orthogonal approach to the functional gradient based ones detailed so far are to treat the problem as a policy learning one. In this approach, we have states corresponding to which predictions have been generated so far, and actions correspond to generating new predictions or outputting final predictions. Examples of this include the value-of-information approach of Gao and Koller [2011], the work on learning a predictor skipping policy of Busa-Fekete et al. [2012], the dynamic predictor re-ordering policy of He et al. [2013], and the object recognition work of Karayev et al. [2012]. In these approaches the policy for selecting which predictions to generate and which features to use is typically generated by modeling the problem as a Markov Decision Process and using some kind of reinforcement learning technique to learn a policy which selects which weak hypotheses or features to compute next.

In the structured setting, Jiang et al. [2012] proposed a technique for reinforcement learning that incorporates a user specified speed/accuracy trade-off distribution, and Weiss and Taskar [2010] proposed a cascaded analog for structured prediction where the solution space is iteratively refined/pruned over time. In contrast, our structured prediction work later in this document is focused on learning a structured predictor with interruptible, anytime properties which is also trained to balance both the structural and feature computation times during the inference procedure. Recent work in computer vision and robotics [Sturgess et al., 2012, de Nijs et al., 2012] has similarly investigated techniques for making approximate inference in graphical models more efficient via a cascaded procedure that iteratively prunes subregions in the scene to analyze.

Previous approaches to the anytime prediction problem have focused on instance-based learning algorithms, such as nearest neighbor classification [Ueno et al., 2006] and novelty detection [Sofman et al., 2010]. These approaches use intelligent instance selection and ordering to acheive rapid performance improvements on common cases, and then typically use the extra time for searching through the 'long tail' of the data distribution and improving result for rare examples. In the case of the latter, the training instances are even dynamically re-ordered based on the distribution of the inputs to the prediction algorithm, further improving performance. As mentioned above, a more recent anytime approach was given by [Xu et al., 2013a], and uses a functional gradient method similar to our initial work in this area [Grubb and Bagnell, 2012].

1.4 Contributions

We now detail the structure of the rest of the document, and outline a number of important contributions made in the various areas related to our anytime prediction approach.

- In Chapter 2 we extend previous work on functional gradient methods and boosting with a framework for analyzing arbitrary convex losses and arbitrary weak learners, as opposed to the classifiers and single output regressors discussed previously. We also analyze the convergence of functional gradient methods for smooth functions, extending previous results and generalizing the notion of weak-to-strong learning to arbitrary weak learners. Finally, we show that the widely used traditional functional gradient approaches fail to converge for non-smooth objective functions, and give algorithms and convergence results that work in the non-smooth setting.

- In Chapter 3 we introduce two extensions to the functional gradient approach detailed in Chapter 2. The first extends functional gradient methods from simple supervised approaches to structured prediction problems using an additive, iterative decoding approach. The second addresses overfitting issues that arise when previous predictions are used as inputs to later predictors in the functional gradient setting, and adapts the method of stacking to this domain to reduce this overfitting.

- In Chapter 4 we detail our analysis of greedy algorithms for the budgeted monotone submodular maximization problem, and derive approximation bounds that demonstrate the near-optimal performance of these greedy approaches. We also extend previous work from the literature with a characterization of approximately submodular functions, and analyze the behvior of algorithms which are approximately greedy as well. Finally, we introduce a modified greedy approach that can achieve good performance for any budget constraint without knowing the budget apriori.

- In Chapter 5 we analyze regularized variants of the sparse approximation problem, and show that this problem is equivalent to the budgeted, approximately submodular setting detailed in Chapter 4. Using these results we derive bounds that show that novel, budgeted or time-aware versions of popular algorithms for this domain are near-optimal as well. In this analysis, we also extend previous algorithms and results for the sparse approximation problem to variants for arbitrary smooth losses and simultaneous targets.

- In Chapter 6 we introduce our cost-greedy, functional gradient approach for solving the anytime prediction. Building on the results in previous chapters, we also show that variants of this anytime prediction approach are guaranteed to have near-optimal performance. Finally, we demonstrate how to extend our anytime prediction approach to a number of applications.

- In Chapter 7 we combine this anytime prediction approach (Chapter 6) with the structured prediction extensions of functional gradient methods (Chapter 3) to obtain an anytime structured prediction algorithm. We then demonstrate this algorithm on the scene understanding domain.

Part I

Functional Gradient Methods

Chapter 2

Functional Gradient Methods

In this chapter we detail our framework for analyzing functional gradient methods, and present convergence results for the general functional gradient approach. Using this new framework we generalize the notion of weak-to-strong learning from the boosting domain to arbitrary weak learners and loss functions. We also extend existing results that give weak-to-strong convergence for smooth losses, and show that for non-smooth losses the widely used standard approach fails, both theoretically and experimentally. To counter this, we develop new algorithms and accompanying convergence results for the non-smooth setting.

2.1 Background

In the functional gradient setting we want to learn a prediction function f which minimizes some objective functional $\mathcal{R}$:

$$\min_f \mathcal{R}[f]. \tag{2.1}$$

We will also assume that f is a linear combination of simpler functions $h \in \mathcal{H}$

$$f(x) = \sum_t \alpha_t h_t(x), \tag{2.2}$$

where $\alpha_t \in \mathbb{R}$. In the boosting literature, these functions $h \in \mathcal{H}$ are typically referred to as *weak predictors* or *weak classifiers* and are some set of functions generated by another learning algorithm which we can easily optimize, known as a *weak learner*. By generating a linear combination of these simpler functions, we hope to obtain better overall performance than any single one of the weak learners h could obtain.

We will now discuss the specific properties of the function space that the functions f and h are drawn from that we will utilize for our analysis, along with various properties of the objective functional $\mathcal{R}$.

Previous work [Mason et al., 1999, Friedman, 2000] has presented the theory underlying function space gradient descent in a variety of ways, but never in a form which is convenient for convergence analysis. Recently, Ratliff [Ratliff, 2009] proposed the L^2 function space as a natural match for this setting. This representation as a vector space is particularly convenient as it dovetails nicely with the analysis of gradient descent based algorithms. We will present here the Hilbert space of functions most relevant to functional gradient boosting.

2.1.1 L^2 Function Space

Given a measurable input set $\mathcal{X}$, a complete vector space $\mathcal{V}$ of outputs, and measure μ over $\mathcal{X}$, the function space $L^2(\mathcal{X}, \mathcal{V}, \mu)$ is the set of all equivalence classes of functions $f : \mathcal{X} \to \mathcal{V}$ such that the Lebesgue integral

$$\int_{\mathcal{X}} \|f(x)\|_{\mathcal{V}}^2 \, d\mu \tag{2.3}$$

is finite. In the special case where μ is a probability measure P with density function $p(x)$, Equation (2.3) is simply equivalent to $\mathbb{E}_P[\|f(x)\|^2]$.

This Hilbert space has a natural inner product and norm:

$$\begin{aligned}
\langle f, g \rangle_\mu &= \int_{\mathcal{X}} \langle f(x), g(x) \rangle_{\mathcal{V}} \, d\mu \\
\|f\|_\mu^2 &= \langle f, f \rangle_\mu \\
&= \int_{\mathcal{X}} \|f(x)\|_{\mathcal{V}}^2 \, d\mu,
\end{aligned}$$

which simplifies as one would expect for the probability measure case:

$$\begin{aligned}
\langle f, g \rangle_P &= \mathbb{E}_P[\langle f(x), g(x) \rangle_{\mathcal{V}}] \\
\|f\|_P^2 &= \mathbb{E}_P[\|f(x)\|_{\mathcal{V}}^2].
\end{aligned}$$

We parameterize these operations by μ to denote their reliance on the underlying measure. For a given input space $\mathcal{X}$ and output space $\mathcal{V}$, different underlying measures can produce a number of spaces over functions $f : \mathcal{X} \to \mathcal{V}$. The underlying measure will also change the elements of the space, which are the equivalence classes for the relation $\sim$:

$$f \sim g \iff \|f - g\|_\mu^2 = 0.$$

The elements of $L^2(\mathcal{X}, \mathcal{V}, \mu)$ are required to be equivalence classes to ensure that the space is a vector space.

In practice, we will often work with the empirical probability distribution $\hat{P}$ for an observed set of points $\{x_n\}_{n=1}^N$. This just causes the vector space operations above to reduce to the empirical

expected values. The inner product becomes

$$\langle f, g \rangle_{\hat{P}} = \frac{1}{N} \sum_{n=1}^{N} \langle f(x_n), g(x_n) \rangle_{\mathcal{V}},$$

and the norm is correspondingly

$$\|f\|_{\hat{P}} = \frac{1}{N} \sum_{n=1}^{N} \|f(x_n)\|_{\mathcal{V}}^2.$$

For the sake of brevity, we will omit the measure μ unless otherwise necessary, as most statements will hold for all measures. When talking about practical implementation, the measure used is assumed to be the empirical probability $\hat{P}$.

2.1.2 Functionals and Convexity

Consider a function space $\mathcal{F} = L^2(\mathcal{X}, \mathcal{V}, \mu)$. We will be analyzing the behavior of functionals $\mathcal{R} : \mathcal{F} \to \mathbb{R}$ over these spaces. A typical example of a functional is the point-wise loss:

$$\mathcal{R}_P[f] = \mathbb{E}_P[\ell(f(x))]. \tag{2.4}$$

To analyze the convergence of functional gradient algorithms across these functionals, we need to rely on a few assumptions. A functional $\mathcal{R}[f]$ is convex if for all $f, g \in \mathcal{F}$ there exists a function $\nabla\mathcal{R}[f]$ such that

$$\mathcal{R}[g] \geq \mathcal{R}[f] + \langle \nabla\mathcal{R}[f], g - f \rangle. \tag{2.5}$$

We say that $\nabla\mathcal{R}[f]$ is a *subgradient* of the functional $\mathcal{R}$ at f. We will write the set of all subgradients, or the subdifferential, of a functional $\mathcal{R}$ at function f as

$$\partial\mathcal{R}[f] = \{\nabla\mathcal{R}[f] \mid \nabla\mathcal{R}[f] \text{ satisfies Equation (2.5) } \forall g\}. \tag{2.6}$$

As an example subgradient, consider the pointwise risk functional given in Equation (2.4). The corresponding subgradient over $L^2(\mathcal{X}, \mathcal{V}, P)$

$$\partial\mathcal{R}_P[f] = \{\nabla \mid \nabla(x) \in \partial\ell(f(x)) \; \forall x \in \text{supp}(P)\} \tag{2.7}$$

where $\partial\ell(f(x))$ is the set of subgradients of the pointwise loss ℓ with respect to the output $f(x)$. For differentiable ℓ, this is just the partial derivative of ℓ with respect to input $f(x)$. Additionally, $\text{supp}(P)$ is the support of measure P, that is, the subset $\mathcal{X}$ such that every open neighborhood of every element $x \in \text{supp}(P)$ has positive measure. This is only necessary to formalize the fact that the subgradient function ∇ need only be defined over elements with positive measure.

To verify this fact, observe that the definition of the subdifferential $\partial\ell(f(x))$ implies that

$$\ell(g(x)) \geq \ell(f(x)) + \langle \nabla(x), g(x) - f(x) \rangle_{\mathcal{V}},$$

for all x with positive measure. Integrating over $\mathcal{X}$ gives

$$\int_{\mathcal{X}} \ell(g(x))p(x)dx \geq \int_{\mathcal{X}} \ell(f(x))p(x)dx + \int_{\mathcal{X}} \langle \nabla(x), g(x) - f(x) \rangle_{\mathcal{V}} p(x)dx,$$

which is exactly the requirement for a subgradient

$$\mathcal{R}_P[g] \geq \mathcal{R}_P[f] + \langle \nabla \mathcal{R}[f], g - f \rangle_P$$

As a special case, we find that the subgradient of the empirical risk $\mathcal{R}_{\hat{P}}[f]$ is simply

$$\partial \mathcal{R}_{\hat{P}}[f] = \{\nabla \mid \nabla(x_n) \in \partial \ell(f(x_n)), n = 1, \ldots, N\}.$$

This function, defined only over the training points x_n, is simply the gradient of the loss ℓ for that point, with respect to the current output $f(x_n)$ at that point.

These subgradients are only valid for the L^2 space corresponding to the particular probability distribution P. In fact, the functional gradient of a risk functional evaluated over a measure P will not always have a defined subgradient in spaces defined using another measure P'. For example no subgradient for the expected loss $\mathcal{R}_P[f]$ exists in the space derived from $\hat{P}$. Similarly, no subgradient of the empirical loss $\mathcal{R}_{\hat{P}}[f]$ exists in the L^2 space derived from the true probability distribution P.

In addition to the simpler notion of convexity, we say that a functional $\mathcal{R}$ is m-*strongly convex* if for all $f, g \in \mathcal{F}$:

$$\mathcal{R}[g] \geq \mathcal{R}[f] + \langle \nabla \mathcal{R}[f], g - f \rangle + \frac{m}{2} \|g - f\|^2 \tag{2.8}$$

for some $m > 0$, and M-*strongly smooth* if

$$\mathcal{R}[g] \leq \mathcal{R}[f] + \langle \nabla \mathcal{R}[f], g - f \rangle + \frac{M}{2} \|g - f\|^2 \tag{2.9}$$

for some $M > 0$.

2.2 Functional Gradient Descent

We now outline the functional gradient-based view of boosting [Mason et al., 1999, Friedman, 2000] and how it relates to other views of boosting. In contrast to the standard gradient descent algorithm, the functional gradient formulation of boosting contains one extra step, where the gradient is not followed directly, but is instead replaced by another vector or function, drawn from the pool of weak predictors $\mathcal{H}$.

From a practical standpoint, a projection step is necessary when optimizing over function space because the functions representing the gradient directly are computationally difficult to manipulate and do not generalize to new inputs well. In terms of the connection to boosting, the allowable search directions $\mathcal{H}$ correspond directly to the set of hypotheses generated by a weak learner.

The functional gradient descent algorithm is given in Algorithm 2.1. Our work in this area addresses two key questions that arise from this view of boosting. First: what are appropriate ways to implement the projection operation? Second: how do we quantify the performance of a given set of weak learners, in general, and how does this performance affect the final performance of the learned function f_t? Conveniently, the function space formalization detailed above gives simple geometric answers to these concerns.

Algorithm 2.1 Functional Gradient Descent

Given: starting point f_0, objective $\mathcal{R}$
for $t = 1, \ldots, T$ **do**
  Compute a subgradient $\nabla_t \in \partial\mathcal{R}[f_{t-1}]$.
  Project ∇_t onto hypothesis space $\mathcal{H}$: $h^* = \text{Proj}\,(\nabla, \mathcal{H})$
  Select a step size α_t.
  Update f: $f_t = f_{t-1} + \alpha_t h^*$.
end for

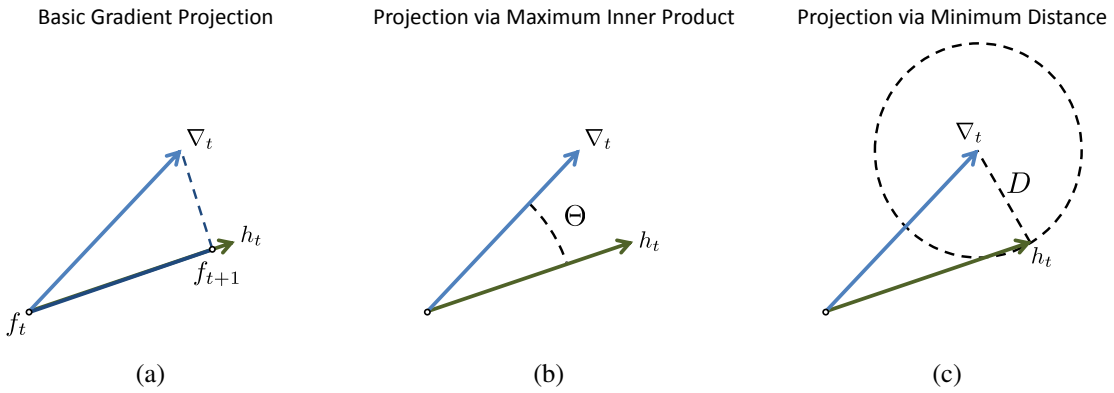


Figure 2.1: Figure demonstrating the geometric intuition underlying (a) the basic gradient projection operation and (b-c) the two methods for optimizing this projection operation over a set of functions $\mathcal{H}$. The inner product formulization (b) minimizes the effective angle between the gradient ∇ and h, while the norm formulization (c) minimizes the effective distance between the two in function space.

For a given a (sub)gradient ∇ and candidate weak learner h, the closest point h' along h can be found using vector projection:

$$h' = \frac{\langle \nabla, h \rangle}{\|h\|^2} h \tag{2.10}$$

Now, given a set of weak learners $\mathcal{H}$ the vector h^* which minimizes the error of the projection

in Equation (2.10) also maximizes the projected length:

$$\begin{aligned} h^* &= \text{Proj}\,(\nabla, \mathcal{H}) \\ &= \arg\max_{h \in \mathcal{H}} \frac{\langle \nabla, h \rangle}{\|h\|}. \end{aligned} \tag{2.11}$$

This is a generalization of the projection operation in Mason et al. [1999] to functions other than classifiers.

For the special case when $\mathcal{H}$ is closed under scalar multiplication, one can instead find h^* by directly minimizing the distance between ∇ and h^*,

$$\begin{aligned} h^* &= \text{Proj}\,(\nabla, \mathcal{H}) \\ &= \arg\min_{h \in \mathcal{H}} \|\nabla - h\|^2 \end{aligned} \tag{2.12}$$

thereby reducing the final projected distance found using Equation (2.10). This projection operation is equivalent to the one given by Friedman [2000], and is suitable for function classes $\mathcal{H}$ that behave like regressors.

A depiction of the geometric intuition behind these projection operations is given in Figure 2.1. These two projection methods provide relatively simple ways to search over any set of allowable directions for the 'best' descent direction. We can also use these same geometric notions to quantify the performance of any given set of weak learners. This guarantee on the performance of each projection step, typically referred to in the traditional boosting literature as the *edge* of a given weak learner set is crucial to our convergence analysis of functional gradient algorithms.

For the projection which maximizes the inner product as in Equation (2.11), we can use the generalized geometric notion of angle to bound performance by requiring that

$$\langle \nabla, h^* \rangle \geq (\cos\theta) \|\nabla\| \|h^*\|,$$

while the equivalent requirement for the norm-based projection in (2.12) is

$$\|\nabla - h^*\|^2 \leq (1 - (\cos\theta)^2) \|\nabla\|^2.$$

It can be seen that this requirement implies the first requirement for arbitrary sets $\mathcal{H}$. In the special case when $\mathcal{H}$ is closed under scalar multiplication, these two requirements are equivalent.

Parameterizing by $\cos\theta$, we can now concisely define the performance potential of a set of weak learners, which will prove useful in later analysis.

Definition 2.2.1. *A set $\mathcal{H}$ has* edge γ *for a given projected gradient ∇ if there exists a vector $h^* \in \mathcal{H}$ such that either $\langle \nabla, h^* \rangle \geq \gamma \|\nabla\| \|h^*\|$ or $\|\nabla - h^*\|^2 \leq (1 - \gamma^2) \|\nabla\|^2$.*

This definition of edge is parameterized by $\gamma \in [0, 1]$, with larger values of edge corresponding to lower projection error and faster algorithm convergence. Historically the edge corresponds to

an increase in performance over some baseline. For instance, in traditional classification problems, the edge corresponds to the edge in performance over random guessing. In our framework, the baseline performer can be thought of as the predictor $h(x) = 0$. The definition of edge given above smoothly interpolates between having no edge over the zero predictor ($\gamma = 0$) and having perfect reconstruction of the projected gradient ($\gamma = 1$).

2.2.1 Relationship to Previous Boosting Work

Though these projection operations apply to any Hilbert space and set $\mathcal{H}$, they also have convenient interpretations when it comes to specific function classes traditionally used as weak learners in boosting.

For a classification-based weak learner with outputs in $\{-1, +1\}$ and an optimization over single output functions $f : \mathcal{X} \to \mathbb{R}$, projecting as in Equation (2.11) is equivalent to solving the weighted classification problem

$$\arg\max_{h \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} |\nabla(x_n)| \mathbb{1} \left(h(x_n) = \operatorname{sgn}(\nabla(x_n)) \right), \tag{2.13}$$

over the training examples x_n, with labels $\operatorname{sgn}(\nabla(x_n))$ and weights $|\nabla(x_n)|$.

For arbitrary real-valued outputs, the projection via norm minimization in Equation (2.12) is equivalent to solving the regression problem

$$\arg\min_{h \in \mathcal{H}} \frac{1}{N} \sum_{n=1}^{N} \|\nabla(x_n) - f(x_n)\|^2$$

again over the training examples x_n with regression targets $\nabla(x_n)$.

Similarly, our notion of weak learner performance in Definition 2.2.1 can be related to previous work. Like our measure of edge, which quantifies performance over the trivial hypothesis $h(x) = 0$, previous work has used similar quantities which capture the advantage over baseline hypotheses.

For weak learners which are binary classifiers, as is the case in AdaBoost [Freund and Schapire, 1997], there is an equivalent notion of edge which refers to the improvement in performance over predicting randomly in the weighted multiclass projection given above. We can show that Definition 2.2.1 is an equivalent measure.

Theorem 2.2.2. *For a weak classifier set $\mathcal{H}$ with outputs in $\{-1, +1\}$ and some gradient ∇, the following statements are equivalent: (1) $\mathcal{H}$ has edge γ for some $\gamma > 0$, and (2) for any non-negative weights w_n over training data x_n, there is a classifier $h \in \mathcal{H}$ which achieves an error of at most $(\frac{1}{2} - \frac{\delta}{2}) \sum_n w_n$ on the weighted classification problem given in Equation (2.13), for some $\delta > 0$.*

Proof. To relate the weighted classification setting and our inner product formulation, let weights $w_n = |\nabla(x_n)|$ and labels $y_n = \text{sgn}(\nabla(x_n))$. We examine classifiers h with outputs in $\{-1, +1\}$.

Consider the AdaBoost weak learner requirement re-written as a sum over the correct examples:

$$\sum_{n,h(x_n)=y_n} w_n \geq (\frac{1}{2} + \frac{\delta}{2}) \sum_n w_n.$$

Breaking the sum over weights into the sum of correct and incorrect weights:

$$\frac{1}{2}(\sum_{n,h(x_n)=y_n} w_n - \sum_{n,h(x_n)\neq y_n} w_n) \geq \frac{\delta}{2} \sum_n w_n.$$

The left hand side of this inequality is just N times the inner product $\langle \nabla, h \rangle$, and the right hand side can be re-written as the 1-norm of the weight vector w, giving:

$$\begin{aligned} N\langle \nabla, h \rangle &\geq \delta \|w\|_1 \\ &\geq \delta \|w\|_2 \end{aligned}$$

Finally, using $\|h\| = 1$ and $\|\nabla\|^2 = \frac{1}{N}\|w\|_2^2$:

$$\langle \nabla, h \rangle \geq \frac{\delta}{\sqrt{N}} \|\nabla\| \|h\|$$

showing that the AdaBoost requirement implies our requirement for edge $\gamma > \frac{\delta}{\sqrt{N}} > 0$.

We can show the converse by starting with our weak learner requirement and expanding:

$$\begin{aligned} \langle \nabla, h \rangle &\geq \gamma \|\nabla\| \|h\| \\ \frac{1}{N}(\sum_{n,h(x_n)=y_n} w_n - \sum_{n,h(x_n)\neq y_n} w_n) &\geq \gamma \|\nabla\| \end{aligned}$$

Then, because $\|\nabla\|^2 = \frac{1}{N}\|w\|_2^2$ and $\|w\|_2 \geq \frac{1}{\sqrt{N}}\|w\|_1$ we get:

$$\begin{aligned} \sum_{n,h(x_n)=y_n} w_n - \sum_{n,h(x_n)\neq y_n} w_n &\geq \gamma \frac{1}{N} \|w\|_1 \\ &\geq \gamma \sum_n w_n \\ \sum_{n,h(x_n)=y_n} w_n &\geq (\frac{1}{2} + \frac{\gamma}{2}) \sum_n w_n, \end{aligned}$$

giving the final AdaBoost edge requirement. ■

In the first part of the previous proof, the scaling of $\frac{1}{\sqrt{N}}$ shows that our implied edge weakens as the number of data points increases in relation to the AdaBoost style edge requirement, an unfortunate but necessary feature. This weakening is necessary because our notion of strong learning is much more general than the previous definitions tailored directly to classification problems and specific loss functions. In those settings, strong learning only guarantees that any dataset can be classified with 0 training error, while our strong learning guarantee gives optimal performance on any convex loss function.

A similar result can be shown for more recent work which generalizes AdaBoost to multiclass classification using multiclass weak learners [Mukherjee and Schapire, 2010]. The notion of edge here uses a cost-sensitive multiclass learning problem as the projection operation, and again the edge is used to compare the performance of the weak learners to that of random guessing. For more details we refer the reader to the work of Mukherjee and Schapire [2010].

In this setting the weak learners h are multiclass classifiers over K outputs, while the comparable weak learners in our functional gradient setting are defined over multiple outputs, $h' : \mathcal{X} \to \mathbb{R}^k$.

Theorem 2.2.3. *For a weak multiclass classifier set $\mathcal{H}$ with outputs in $\{1, \ldots, K\}$, let the modified hypothesis space $\mathcal{H}'$ contain a hypothesis $h' : \mathcal{X} \to \mathbb{R}^K$ for each $h \in \mathcal{H}$ such that $h'(x)_k = 1$ if $h(x) = k$ and $h'(x) = -\frac{1}{K-1}$ otherwise. Then, for a given gradient function ∇, the following statements are equivalent: (1) $\mathcal{H}'$ has edge γ for some $\gamma > 0$, and (2) $\mathcal{H}$ satisfies the performance over baseline requirements detailed in Theorem 1 of [Mukherjee and Schapire, 2010].*

Proof. In this section we consider the multiclass extension of the previous setting. Instead of a weight vector we now have a matrix of weights w where w_{nk} is the weight or reward for classifying example x_n as class k. We can simply let weights $w_{nk} = \nabla(x_{nk})$ and use the same weak learning approach as in [Mukherjee and Schapire, 2010]. Given classifiers $h(x)$ which output a label in $\{1, \ldots, K\}$, we convert to an appropriate weak learner for our setting by building a function $h'(x)$ which outputs a vector $y \in \mathcal{R}^K$ such that $y_k = 1$ if $h(x) = k$ and $y_k = -\frac{1}{K-1}$ otherwise.

The equivalent AdaBoost style requirement uses costs $c_{nk} = -w_{nk}$ and minimizes instead of maximizing, but here we state the weight or reward version of the requirement. More details on this setting can be found in [Mukherjee and Schapire, 2010]. We also make the additional assumption that $\sum_k w_{nk} = 0, \forall n$ without loss of generality. This assumption is fine as we can take a given weight matrix w and modify each row so it has 0 mean, and still have a valid classification matrix as per [Mukherjee and Schapire, 2010]. Furthermore, this modification does not affect the edge over random performance of a multiclass classifier under their framework.

Again consider the multiclass AdaBoost weak learner requirement re-written as a sum of the

weights over the predicted class for each example:

$$\sum_n w_{nh(x_n)} \geq (\frac{1}{K} - \frac{\delta}{K}) \sum_{n,k} w_{nk} + \delta \sum_n w_{ny_n}$$

we can then convert the sum over correct labels to the max-norm on weights and multiply through by $\frac{K}{K-1}$:

$$\begin{aligned}
\sum_n w_{nh(x_n)} &\geq \frac{1}{K} \sum_{n,k} w_{nk} - \frac{\delta}{K} \sum_{n,k} w_{nk} + \delta \sum_n w_{ny_n} \\
\frac{K}{K-1} \sum_n w_{nh(x_n)} &\geq \frac{1}{K-1} \sum_{n,k} w_{nk} + \frac{K}{K-1} (\delta \sum_n \|w_n\|_\infty - \frac{\delta}{K} \sum_{n,k} w_{nk}) \\
\frac{K}{K-1} \sum_n w_{nh(x_n)} - \frac{1}{K-1} \sum_{n,k} w_{nk} &\geq \frac{K}{K-1} (\delta \sum_n \|w_n\|_\infty - \frac{\delta}{K} \sum_{n,k} w_{nk})
\end{aligned}$$

by the fact that the correct label $y_n = \arg\max_k w_{nk}$.

The left hand side of this inequality is just the function space inner product:

$$N\langle \nabla, h' \rangle \geq \frac{K}{K-1} (\delta \sum_n \|w_n\|_\infty - \frac{\delta}{K} \sum_{n,k} w_{nk}).$$

Using the fact that $\sum_k w_{nk} = 0$ along with $\|\nabla\| \leq \frac{1}{\sqrt{N}} \sum_n \|w_n\|_2$ and $\|h'\| = \sqrt{\frac{K}{K-1}}$ we can now bound the right hand side:

$$\begin{aligned}
N\langle \nabla, h' \rangle &\geq \frac{K}{K-1} \delta \sum_n \|w_n\|_\infty \\
&\geq \frac{K}{K-1} \delta \sum_n \|w_n\|_2 \\
&\geq \frac{K}{K-1} \delta \sqrt{N} \|\nabla\| \\
&\geq \sqrt{\frac{K}{K-1}} \delta \sqrt{N} \|\nabla\| \|h'\| \\
\langle \nabla, h \rangle &\geq \sqrt{\frac{K}{K-1}} \delta \frac{1}{\sqrt{N}} \|\nabla\| \|h'\|
\end{aligned}$$

For $K \geq 2$ we get $\gamma \geq \frac{\delta}{\sqrt{N}}$, showing that the existence of the AdaBoost style edge implies the existence of ours. Again, while the requirements are equivalent for some fixed dataset, we

see a weakening of the implication as the dataset grows large, an unfortunate consequence of our broader strong learning goals.

Now to show the other direction, start with the inner product formulation:

$$\begin{aligned}
\langle \nabla, h' \rangle &\geq \delta \|\nabla\| \|h'\| \\
\frac{1}{N}(\sum_n w_{nh(x_n)} - \frac{1}{K-1} \sum_{n,k \neq h(x_n)} w_{nk}) &\geq \delta \|\nabla\| \|h'\| \\
\frac{1}{N}(\frac{K}{K-1} \sum_n w_{nh(x_n)} - \frac{1}{K-1} \sum_{n,k} w_{nk}) &\geq \delta \|\nabla\| \|h'\|
\end{aligned}$$

Using $\|h'\| = \sqrt{\frac{K}{K-1}}$ and $\|\nabla\| \geq \frac{1}{N} \sum_n \|w_n\|_2$ we can show:

$$\frac{K}{K-1} \sum_n w_{nh(x_n)} - \frac{1}{K-1} \sum_{n,k} w_{nk} \geq \delta \sum_n \|w_n\|_2 \sqrt{\frac{K}{K-1}}.$$

Rearranging we get:

$$\begin{aligned}
\frac{K}{K-1} \sum_n w_{nh(x_n)} &\geq \frac{1}{K-1} \sum_{n,k} w_{nk} + \delta \sum_n \|w_n\|_2 \sqrt{\frac{K}{K-1}} \\
\sum_n w_{nh(x_n)} &\geq \frac{1}{K} \sum_{n,k} w_{nk} + \frac{K-1}{K} \sqrt{\frac{K}{K-1}} \delta \sum_n \|w_n\|_2 \\
\sum_n w_{nh(x_n)} &\geq \frac{1}{K} \sum_{n,k} w_{nk} + \sqrt{\frac{K}{K-1}} \delta (\sum_n \|w_n\|_2 - \frac{1}{K} \sum_n \|w_n\|_2)
\end{aligned}$$

Next, bound the 2-norms using $\|w_n\|_2 \geq \frac{1}{\sqrt{K}} \|w_n\|_1$ and $\|w_n\|_2 \geq \|w_n\|_\infty$ and then rewrite as sums of corresponding weights to show the multiclass AdaBoost requirement holds:

$$\begin{aligned}
\sum_n w_{nh(x_n)} &\geq (\frac{1}{K} - \frac{\delta}{\sqrt{K-1}K}) \sum_{n,k} w_{nk} + \sqrt{\frac{K}{K-1}} \delta \sum_n \|w_n\|_\infty \\
\sum_n w_{nh(x_n)} &\geq (\frac{1}{K} - \frac{\delta}{K}) \sum_{n,k} w_{nk} + \delta \sum_n w_{ny_n}
\end{aligned}$$

■

2.3 Restricted Gradient Descent

We will now focus on using the machinery developed above to analyze the behavior of variants of the functional gradient boosting method on problems of the form:

$$\min_{f \in \mathcal{F}} \mathcal{R}[f],$$

where f is a sum of weak learners taken from some set $\mathcal{H} \subset \mathcal{F}$.

In line with previous boosting work, we will specifically consider cases where the edge requirement in Definition 2.2.1 is met for some γ at every iteration, and seek convergence results where the empirical risk $\mathcal{R}_{\hat{P}}[f_t]$ approaches the optimal training performance $\min_{f \in \mathcal{F}} \mathcal{R}_{\hat{P}}[f]$. For the rest of the work it is assumed that function space operations and functionals are evaluated with respect to the empirical distribution $\hat{P}$. This work does not attempt to analyze the convergence of the true risk, or generalization error, $\mathcal{R}_P[f]$.

Algorithm 2.2 Basic Gradient Projection Algorithm

Given: starting point f_0, objective $\mathcal{R}$, step size schedule $\{\eta_t\}_{t=1}^T$
for $t = 1, \ldots, T$ **do**
  Compute a subgradient $\nabla_t \in \partial \mathcal{R}[f_{t-1}]$.
  Compute $h^* = \text{Proj}\,(\nabla, \mathcal{H})$.
  Update f: $f_t = f_{t-1} - \eta_t \frac{\langle h^*, \nabla_t \rangle}{\|h^*\|^2} h^*$.
end for

In order to complete this analysis, we will consider a general version of the functional gradient boosting procedure given in Algorithm 2.1 which we call *restricted gradient descent*. While we will continue to use the notation of L^2 function spaces specifically, the convergence analysis presented can be applied generally to any Hilbert space.

Let $\mathcal{F}$ be a Hilbert space and $\mathcal{H} \subset \mathcal{F}$ be a set of allowable search directions, or *restriction set*. This set, which in traditional boosting is the set of weak learners, can also be though of as a basis for the subset of $\mathcal{F}$ that we are actually searching over.

In the restricted gradient descent setting we want to perform a gradient descent-like procedure, while only every taking steps along search directions drawn from $\mathcal{H}$. To do this, we will project each gradient on to the set $\mathcal{H}$ as in functional gradient boosting. Algorithm 2.2 gives the basic algorithm for projecting the gradients and taking steps. The main difference between this algorithm and the previous functional gradient one is the extra term in the actual (projected) gradient step which depends on $\langle \nabla, h^* \rangle$. Otherwise this algorithm is functionally the same as the functional gradient boosting method.

Now, using the definition of edge given in Definition 2.2.1, we will first analyze the performance of this restricted gradient descent algorithm on smooth functionals.

2.4 Smooth Convergence Results

With respect to the functional gradient boosting literature, an earlier result showing $O((1-\frac{1}{C})^T)$ convergence of the objective to optimality for smooth functionals is given by Rätsch et al. [2002] using results from the optimization literature on coordinate descent. Alternatively, this gives a $O(\log(\frac{1}{\epsilon}))$ result for the number of iterations required to achieve error ϵ. Similar to our result, this work relies on the smoothness of the objective as well as the weak learner performance, but uses the more restrictive notion of edge from previous boosting literature specifically tailored to PAC weak learners (classifiers). This previous result also has an additional dependence on the number of weak learners and number of training examples.

We will now give a generalization of the result in Rätsch et al. [2002] which uses our more general definition of weak learner edge. This result also relates to the previous work of Mason et al. [1999]. In that work, a similar convergence analysis is given, but the analysis only states that, under similar conditions, the gradient boosting procedure will eventually converge. Our analysis, however, considers the speed of convergence and the impact that our definition of weak learner edge has on the convergence.

Recall the strong smoothness and strong convexity properties given earlier in Equations (2.9) and (2.8). Using these two properties, we can now derive a convergence result for unconstrained optimization over smooth functions.

Theorem 2.4.1 (Generalization of Theorem 4 in [Rätsch et al., 2002])**.** *Let $\mathcal{R}$ be a m-strongly convex and M-strongly smooth functional over $\mathcal{F}$. Let $\mathcal{H} \subset \mathcal{F}$ be a restriction set with edge γ for every gradient ∇_t that is projected on to $\mathcal{H}$. Let $f^* = \arg\min_{f\in\mathcal{F}} \mathcal{R}[f]$. Given a starting point f_0 and step size $\eta_t = \frac{1}{M}$, after T iterations of Algorithm 2.2 we have:*

$$\mathcal{R}[f_T] - \mathcal{R}[f^*] \leq (1 - \frac{\gamma^2 m}{M})^T (\mathcal{R}[f_0] - \mathcal{R}[f^*]).$$

Proof. Starting with the definition of strong smoothness, and examining the objective value at time $t+1$ we have:

$$\mathcal{R}[f_{t+1}] \leq \mathcal{R}[f_t] + \langle \nabla\mathcal{R}[f_t], f_{t+1} - f_t \rangle + \frac{M}{2}\|f_{t+1} - f_t\|^2$$

Then, using $f_{t+1} = f_t + \frac{1}{M}\frac{\langle \nabla\mathcal{R}[f_t], h_t\rangle}{\|h_t\|^2} h_t$ we get:

$$\mathcal{R}[f_{t+1}] \leq \mathcal{R}[f_t] - \frac{1}{2M}\frac{\langle \nabla\mathcal{R}[f_t], h_t\rangle^2}{\|h_t\|^2}$$

Subtracting the optimal value from both sides and applying the edge requirement we get:

$$\mathcal{R}[f_{t+1}] - \mathcal{R}[f^*] \leq \mathcal{R}[f_t] - \mathcal{R}[f^*] - \frac{\gamma^2}{2M}\|\nabla\mathcal{R}[f_t]\|^2$$

From the definition of strong convexity we know $\|\nabla\mathcal{R}[f_t]\|^2 \geq 2m(\mathcal{R}[f_t] - \mathcal{R}[f^*])$ where f^* is the minimum point. Rearranging we can conclude that:

$$\mathcal{R}[f_{t+1}] - \mathcal{R}[f^*] \leq (\mathcal{R}[f_t] - \mathcal{R}[f^*])(1 - \frac{\gamma^2 m}{M})$$

Recursively applying the above bound starting at $t = 0$ gives the final bound on $\mathcal{R}[f_T] - \mathcal{R}[f_0]$. ■

The result above holds for the fixed step size $\frac{1}{M}$ as well as for step sizes found using a line search along the descent direction, as they will only improve the convergence rate, because we are considering the convergence at each iteration independently in the above proof.

Theorem 2.4.1 gives, for strongly smooth objective functionals, a convergence rate of $O((1 - \frac{\gamma^2 m}{M})^T)$. This is very similar to the $O((1-4\gamma^2)^{\frac{T}{2}})$ convergence of AdaBoost [Freund and Schapire, 1997], or $O((1 - \frac{1}{C})^T)$ convergence rate given by Rätsch et al. [2002], as all require $O(\log(\frac{1}{\epsilon}))$ iterations to get performance within ϵ of the optimal result.

While the AdaBoost result generally provides tighter bounds, this relatively naive method of gradient projection is able to obtain reasonably competitive convergence results while being applicable to a much wider range of problems. This is expected, as the proposed method derives no benefit from loss-specific optimizations and can use a much broader class of weak learners. This comparison is a common scenario within optimization: while highly specialized algorithms can often perform better on specific problems, general solutions often obtain equally impressive results, albeit less efficiently, while requiring much less effort to implement.

2.4.1 Non-smooth Degeneration

Unfortunately, the naive approach to restricted gradient descent breaks down quickly in more general cases such as non-smooth objectives. Consider the following example objective (also depicted in Figure 2.2 over two points x_1, x_2: $\mathcal{R}[f] = 2|f(x_1)| + |f(x_2)|$. For this problem, a valid subgradient is ∇ such that $\nabla(x_1) = 2\,\mathrm{sgn}(x_1)$ and $\nabla(x_2) = \mathrm{sgn}(x_2)$. We will assume that $\mathrm{sgn}(0) = 1$, to give us a unique subgradient for the $x_1 = 0$ or $x_2 = 0$ case.

Now consider the hypothesis set $h \in \mathcal{H}$ such that either $h(x_1) \in \{-1, +1\}$ and $h(x_2) = 0$, or $h(x_1) = 0$ and $h(x_2) \in \{-1, +1\}$. The algorithm will always select h^* such that $h^*(x_2) = 0$ when projecting gradients from the example objective, and the set $\mathcal{H}$ will always have reasonably large edge with respect to ∇.

This procedure, however, gives a final function with perfect performance on x_1 and arbitrarily poor unchanged performance on x_2, depending on choice of starting function f_0. Even if the loss

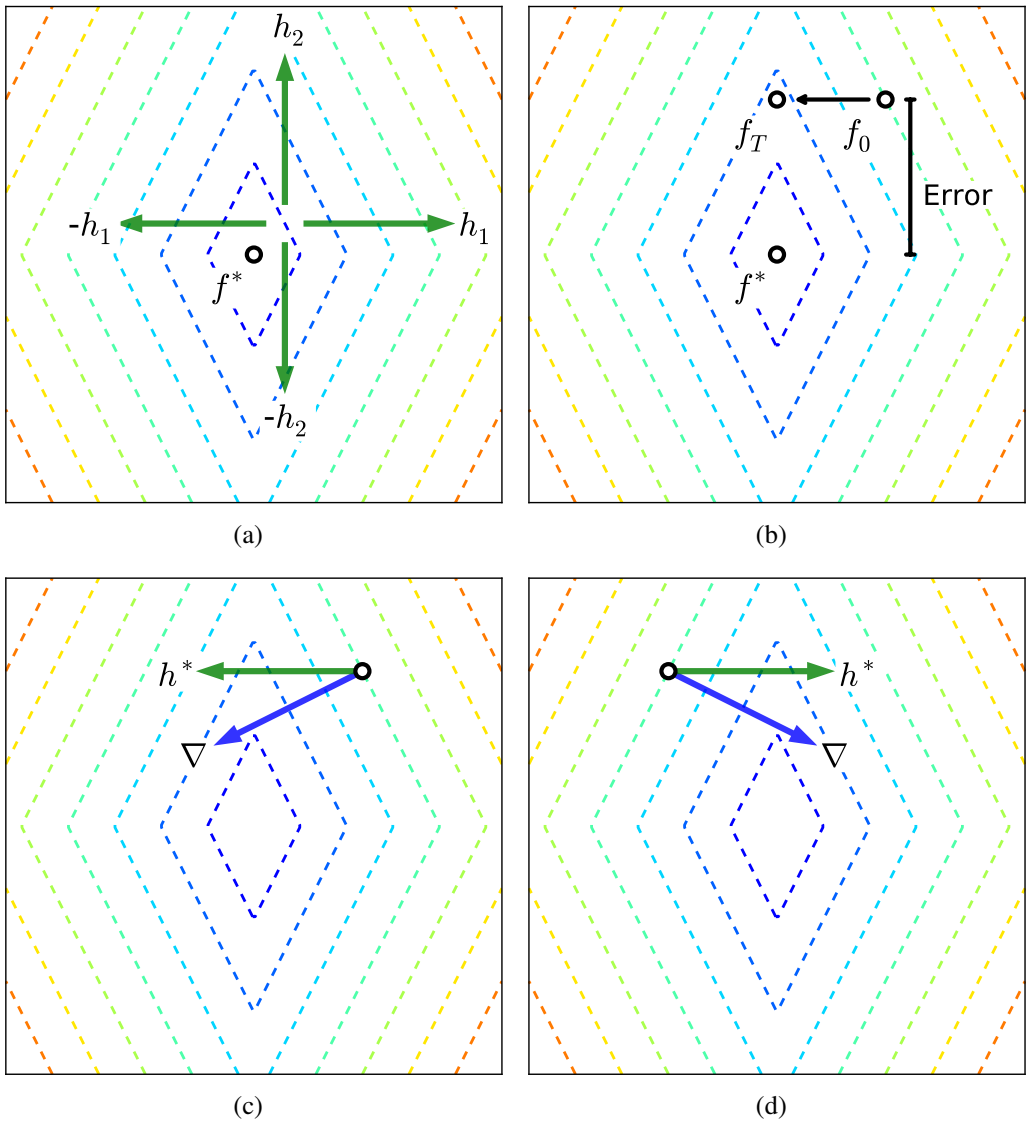


Figure 2.2: A demonstration of a non-smooth objective for which the basic restricted gradient descent algorithm fails. The possible weak predictors and optimal value f^* are depicted in (a), while (b) gives the result of running the basic restricted gradient algorithm on this problem for an example starting point f_0 and demonstrates the optimality gap. This gap is due to the fact that the algorithm will only ever select h_1 or $-h_1$ as possible descent directions, as depicted in (c) and (d).

on training point x_2 is substantial due to a bad starting location, naively applying the basic gradient projection algorithm will not correct it.

An algorithm which greedily projects subgradients of $\mathcal{R}$, such as Algorithm 2.2, will not be able to obtain strong performance results for cases like these. The algorithms in the next section

overcome this obstacle by projecting modified versions of the subgradients of the objective at each iteration.

2.5 General Convex Convergence Results

For the convergence analysis of general convex functions we now switch to analyzing the average optimality gap:

$$\frac{1}{T}\sum_{t=1}^{T}[\mathcal{R}[f_t] - \mathcal{R}[f^*]],$$

where $f^* = \arg\min_{f \in \mathcal{F}} \sum_{t=1}^{T} \mathcal{R}[f]$ is the fixed hypothesis which minimizes loss.

By showing that the average optimality gap approaches 0 as T grows large, for decreasing step sizes, it can be shown that the optimality gap $\mathcal{R}[f_t] - \mathcal{R}[f^*]$ also approaches 0.

This analysis is similar to the standard no-regret online learning approach, but we restrict our analysis to the case when $\mathcal{R}_t = \mathcal{R}$. This is because the true online setting typically involves receiving a new dataset at every time t, and hence a different data distribution $\hat{P}_t$, effectively changing the underlying L^2 function space of operations such as gradient projection at every time step, making comparison of quantities at different time steps difficult in the analysis. The convergence analysis for the online case is beyond the scope of our work and is not presented here.

The convergence results to follow are similar to previous convergence results for the standard gradient descent setting [Zinkevich, 2003, Hazan et al., 2006], but with a number of additional error terms due to the gradient projection step. Sutskever [2009] has previously studied the convergence of gradient descent with gradient projection errors using an algorithm similar to Algorithm 2.2, but the analysis does not focus on the weak to strong learning guarantee we seek. [1] In order to obtain this guarantee we now present two new algorithms.

Our first general convex solution, shown in Algorithm 2.3, overcomes this issue by using a meta-boosting strategy. At each iteration t instead of projecting the gradient ∇_t onto a single hypothesis h^*, we use the naive algorithm to construct h^* out of a small number of restricted steps, optimizing over the distance $\|\nabla_t - h^*\|^2$. By increasing the number of weak learners trained at each iteration over time, we effectively decrease the gradient projection error at each iteration. As the average projection error approaches 0, the performance of the combined hypothesis approaches optimal. We can now prove convergence results for this algorithm for both strongly convex and convex functionals.

Theorem 2.5.1. *Let $\mathcal{R}$ be a m-strongly convex functional over $\mathcal{F}$. Let $\mathcal{H} \subset \mathcal{F}$ be a re-*

[1] In fact, Sutskever's convergence results can be used to show that the bound on training error for the basic gradient projection algorithm asymptotically approaches the average error of the weak learners, only indicating that you are guaranteed to find a hypothesis which does no worse than any individual weak learner, despite its increased complexity.

Algorithm 2.3 Repeated Gradient Projection Algorithm

Given: starting point f_0, objective $\mathcal{R}$, step size schedule $\{\eta_t\}_{t=1}^T$
for $t = 1, \ldots, T$ **do**
  Compute subgradient $\nabla_t \in \partial\mathcal{R}[f_{t-1}]$.
  Let $\nabla' = \nabla_t$, $h^* = 0$.
  for $k = 1, \ldots, t$ **do**
    Compute $h_k^* = \text{Proj}\,(\nabla', \mathcal{H})$.
    $h^* \leftarrow h^* + \frac{\langle h_k^*, \nabla' \rangle}{\|h_k^*\|^2} h_k^*$.
    $\nabla' \leftarrow \nabla' - h_k^*$.
  end for
  Update f: $f_t = f_{t-1} - \eta_t h^*$.
end for

striction set with edge γ for every ∇' that is projected on to $\mathcal{H}$. Let $\|\nabla\mathcal{R}[f]\| \leq G$. Let $f^* = \arg\min_{f\in\mathcal{F}}\mathcal{R}[f]$. Given a starting point f_0 and step size $\eta_t = \frac{5}{4mt}$, after T iterations of Algorithm 2.3 we have:

$$\frac{1}{T}\sum_{t=1}^{T}[\mathcal{R}[f_t] - \mathcal{R}[f^*]] \leq \frac{5G^2}{2mT}(1 + \ln T + \frac{1-\gamma^2}{\gamma^2}).$$

Proof. First, we start by bounding the potential $\|f_t - f^*\|^2$, similar to the potential function arguments of Zinkevich [2003] and Hazan et al. [2006], but with a different descent step:

$$\begin{aligned}
\|f_{t+1} - f^*\|^2 &\leq \|f_t - \eta_{t+1}(h_t) - f^*\|^2 \\
&= \|f_t - f^*\|^2 + \eta_{t+1}^2\|h_t\|^2 - 2\eta_{t+1}\langle f_t - f^*, h_t - \nabla_t\rangle - 2\eta_{t+1}\langle f_t - f^*, \nabla_t\rangle \\
\langle f^* - f_t, \nabla_t\rangle &\geq \frac{1}{2\eta_{t+1}}\|f_{t+1} - f^*\|^2 - \frac{1}{2\eta_{t+1}}\|f_t - f^*\|^2 - \frac{\eta_{t+1}}{2}\|h_t\|^2 - \langle f^* - f_t, h_t - \nabla_t\rangle
\end{aligned}$$

Using the definition of strong convexity and summing:

$$\begin{aligned}\sum_{t=1}^{T}\mathcal{R}[f^*] &\geq \sum_{t=1}^{T}\mathcal{R}[f_t] + \sum_{t=1}^{T}\langle f^* - f_t, \nabla_t\rangle + \sum_{t=1}^{T}\frac{m}{2}\|f^* - f_t\|^2 \\ &\geq \sum_{t=1}^{T}\mathcal{R}[f_t] + \sum_{t=1}^{T}\frac{1}{2}\|f_t - f^*\|^2(\frac{1}{\eta_t} - \frac{1}{\eta_{t+1}} + m) \\ &\quad - \sum_{t=1}^{T}\frac{\eta_{t+1}}{2}\|h_t\|^2 - \sum_{t=1}^{T}\langle f^* - f_t, h_t - \nabla_t\rangle\end{aligned}$$

Setting $\eta_t = \frac{5}{4mt}$ and use bound $\|h_t\| \leq 2\|\nabla_t\| \leq 2G$:

$$\begin{aligned}\sum_{t=1}^{T}\mathcal{R}[f^*] &\geq \sum_{t=1}^{T}\mathcal{R}[f_t] + \sum_{t=1}^{T}\frac{m}{2}\frac{1}{5}\|f_t - f^*\|^2 - \sum_{t=1}^{T}\frac{5}{8mt}\|h_t\|^2 - \sum_{t=1}^{T}\langle f^* - f_t, h_t - \nabla_t\rangle \\ &\geq \sum_{t=1}^{T}\mathcal{R}[f_t] - \frac{5G^2}{2m}\sum_{t=1}^{T}\frac{1}{t} - \sum_{t=1}^{T}\left[\langle f^* - f_t, h_t - \nabla_t\rangle) - \frac{m}{2}\frac{1}{5}\|f^* - f_t\|^2\right] \\ &\geq \sum_{t=1}^{T}\mathcal{R}[f_t] - \frac{5G^2}{2m}(1 + \ln T) - \sum_{t=1}^{T}\left[\langle f^* - f_t, h_t - \nabla_t\rangle) - \frac{m}{2}\frac{1}{5}\|f^* - f_t\|^2\right]\end{aligned}$$

Next we bound the remaining term by using a variant of the Polarization identity. First we expand

$$\frac{1}{2}\left\|\sqrt{c}A - \frac{1}{\sqrt{c}}B\right\|^2 = \frac{c}{2}\|A\|^2 + \frac{1}{2c}\|B\|^2 - \langle A, B\rangle.$$

Then, using the fact that $\left\|\sqrt{c}A - \frac{1}{\sqrt{c}}B\right\|^2 \geq 0$, we can bound:

$$\frac{1}{2c}\|B\|^2 \geq \langle A, B\rangle - \frac{c}{2}\|A\|^2.$$

Using that bound and the result from Theorem 2.4.1 we can bound the error at each step t:

$$\begin{aligned}\sum_{t=1}^{T}\mathcal{R}[f^*] &\geq \sum_{t=1}^{T}\mathcal{R}[f_t] - \frac{5G^2}{2m}(1+\ln T) - \sum_{t=1}^{T}\frac{5}{2m}\|h_t - \nabla_t\|^2 \\ &\geq \sum_{t=1}^{T}\mathcal{R}[f_t] - \frac{5G^2}{2m}(1+\ln T) - \frac{5G^2}{2m}\sum_{t=1}^{T}(1-\gamma^2)^t \\ &\geq \sum_{t=1}^{T}\mathcal{R}[f_t] - \frac{5G^2}{2m}(1+\ln T) - \frac{5G^2}{2m}\frac{1-\gamma^2}{\gamma^2}\end{aligned}$$

giving the final bound.

This bound can be improved slightly by instead using the step size $\eta_t = \frac{\lambda}{mt}$, in which case the final bound will be

$$\sum_{t=1}^{T}\mathcal{R}[f^*] \geq \sum_{t=1}^{T}\mathcal{R}[f_t] - \frac{2G^2\lambda}{m}(1+\ln T) - \frac{G^2}{2m}\frac{\lambda}{\lambda-1}\frac{1-\gamma^2}{\gamma^2}$$

Minimizing this over λ gives the optimal value step of

$$\lambda = \sqrt{\frac{\frac{1-\gamma^2}{\gamma^2}}{4(1+\ln T)}} + 1.$$

■

The proof relies on the fact that as the number of iterations increases, our gradient projection error approaches 0 at the rate given in Theorem 2.4.1, causing the behavior of Algorithm 2.3 to approach the standard gradient descent algorithm. The additional error term in the result is a bound on the geometric series describing the errors introduced at each time step.

Theorem 2.5.2. *Let $\mathcal{R}$ be a convex functional over $\mathcal{F}$. Let $\mathcal{H} \subset \mathcal{F}$ be a restriction set with edge γ for every ∇' that is projected on to $\mathcal{H}$. Let $\|\nabla\mathcal{R}[f]\| \leq G$ and $\|f\| \leq F$ for all $f \in \mathcal{F}$. Let $f^* = \arg\min_{f\in\mathcal{F}}\mathcal{R}[f]$. Given a starting point f_0 and step size $\eta_t = \frac{1}{\sqrt{t}}$, after T iterations of Algorithm 2.3 we have:*

$$\frac{1}{T}\sum_{t=1}^{T}[\mathcal{R}[f_t] - \mathcal{R}[f^*]] \leq \frac{F^2}{\sqrt{T}} + \frac{2G^2}{\sqrt{T}} + 2FG\frac{1-\gamma^2}{\gamma^2 T}.$$

Proof. Like the last proof, we start with the altered potential and sum over the definition of

convexity:

$$\sum_{t=1}^{T} \mathcal{R}[f^*] \geq \sum_{t=1}^{T} \mathcal{R}[f_t] + \sum_{t=1}^{T} \frac{1}{2}\|f_t - f^*\|^2 (\frac{1}{\eta_t} - \frac{1}{\eta_{t+1}} + m) - \sum_{t=1}^{T} \frac{\eta_{t+1}}{2}\|h_t\|^2 - \sum_{t=1}^{T} \langle f^* - f_t, h_t - \nabla_t \rangle$$

Setting $\eta_t = \frac{1}{\sqrt{t}}$ and using bound $\|h_t\| \leq 2\|\nabla_t\| \leq 2G$ and the result from Theorem 2.4.1 we can bound the error at each step t:

$$\begin{aligned}
\sum_{t=1}^{T} \mathcal{R}[f^*] &\geq \sum_{t=1}^{T} \mathcal{R}[f_t] - \frac{1}{\eta_T}\|f_T - f^*\|^2 - 2G^2 \sum_{t=1}^{T} \frac{1}{\sqrt{t}} - \sum_{t=1}^{T} \langle f^* - f_t, h_t - \nabla_t \rangle \\
&\geq \sum_{t=1}^{T} \mathcal{R}[f_t] - F^2\sqrt{T} - 2G^2\sqrt{T} - FG \sum_{t=1}^{T} \sqrt{(1-\gamma^2)^t} \\
&\geq \sum_{t=1}^{T} \mathcal{R}[f_t] - F^2\sqrt{T} - 2G^2\sqrt{T} - 2FG\frac{1-\gamma^2}{\gamma^2}
\end{aligned}$$

giving the final bound. ■

Again, the result is similar to the standard gradient descent result, with an added error term dependent on the edge γ.

In practice, the large number of weak learners trained using this method could become prohibitive, and the performance of this algorithm in practice is often much better than that given by the derived bounds above.

In this case, an alternative version of the repeated projection algorithm allows for a variable number of weak learners to be trained at each iteration. An accuracy threshold for each gradient projection could be derived given a desired accuracy for the final hypothesis, and this threshold can be used to train weak learners at each iteration until the desired accuracy is reached. In practice, this would allow for only the number of weak learners required to reach a given accuracy target to be trained, reducing the total number of weak learners.

Algorithm 2.4 gives another approach for optimizing over convex objectives which may also address this issue of the increasingly large number of weak learners. Like the previous approach, the projection error at each time step is used again in projection, but a new step is not taken immediately to decrease the projection error. Instead, this approach keeps track of the residual error left over after projection and includes this error in the next projection step. This forces the projection steps to eventually account for past errors, preventing the possibility of systematic error being adversarially introduced through the weak learner set.

As with Algorithm 2.3, we can derive similar convergence results for strongly-convex and

Algorithm 2.4 Residual Gradient Projection Algorithm

Given: starting point f_0, objective $\mathcal{R}$, step size schedule $\{\eta_t\}_{t=1}^T$
Let $\Delta = 0$.
for $t = 1, \ldots, T$ **do**
  Compute subgradient $\nabla_t \in \partial\mathcal{R}[f_{t-1}]$.
  $\Delta = \Delta + \nabla_t$.
  Compute $h^* = \text{Proj}\,(\Delta, \mathcal{H})$.
  Update f: $f_t = f_{t-1} - \eta_t \frac{\langle h^*, \Delta\rangle}{\|h^*\|^2} h^*$.
  Update residual: $\Delta = \Delta - \frac{\langle h^*, \Delta\rangle}{\|h^*\|^2} h^*$
end for

general convex functionals for this new residual-based algorithm.

Theorem 2.5.3. *Let $\mathcal{R}$ be a m-strongly convex functional over $\mathcal{F}$. Let $\mathcal{H} \subset \mathcal{F}$ be a restriction set with edge γ for every Δ that is projected on to $\mathcal{H}$. Let $\|\nabla\mathcal{R}[f]\| \leq G$. Let $f^* = \arg\min_{f\in\mathcal{F}}\mathcal{R}[f]$. Let $c = \frac{2}{\gamma^2}$. Given a starting point f_0 and step size $\eta_t = \frac{1}{mt}$, after T iterations of Algorithm 2.4 we have:*

$$\frac{1}{T}\sum_{t=1}^{T}[\mathcal{R}[f_t] - \mathcal{R}[f^*]] \leq \frac{2c^2G^2}{mT}(1 + \ln T + \frac{2}{T}).$$

Proof. Like the proof of Theorem 2.5.1, we again use a potential function and sum over the definition of convexity:

$$\begin{aligned}
\sum_{t=1}^{T}\mathcal{R}[f^*] \geq & \sum_{t=1}^{T}\mathcal{R}[f_t] + \sum_{t=1}^{T}\frac{1}{2}\|f_t - f^*\|^2(\frac{1}{\eta_t} - \frac{1}{\eta_{t+1}} + m) - \\
& \sum_{t=1}^{T}\frac{\eta_{t+1}}{2}\|h_t\|^2 - \sum_{t=1}^{T}\langle f^* - f_t, h_t - (\Delta_t + \nabla_t)\rangle - \sum_{t=0}^{T-1}\langle f^* - f_{t+1}, \Delta_{t+1}\rangle \\
\geq & \sum_{t=1}^{T}\mathcal{R}[f_t] + \sum_{t=1}^{T}\frac{1}{2}\|f_t - f^*\|^2(\frac{1}{\eta_t} - \frac{1}{\eta_{t+1}} + m) - \\
& \sum_{t=1}^{T}\frac{\eta_{t+1}}{2}\|h_t\|^2 - \sum_{t=1}^{T}\langle f^* - f_t, h_t - (\Delta_t + \nabla_t)\rangle - \sum_{t=0}^{T-1}\langle f^* - f_t, \Delta_{t+1}\rangle - \sum_{t=0}^{T-1}\langle \eta_{t+1}h_t, \Delta_{t+1}\rangle
\end{aligned}$$

where h_t is the augmented step taken in Algorithm 2.4.

Setting $\eta_t = \frac{1}{\gamma t}$ and use bound $\|h_t\| \leq \|\nabla_t\| \leq G$, along with $\Delta_{t+1} = (\Delta_t + \nabla_t) - h_t$:

$$\sum_{t=1}^{T} \mathcal{R}[f^*] \geq \sum_{t=1}^{T} \mathcal{R}[f_t] \sum_{t=1}^{T} \frac{\eta_{t+1}}{2}\|h_t\|^2 - (\langle f^* - f_{T+1}, \Delta_{t+1}\rangle - \frac{mT}{2}\|f^* - f_{T+1}\|^2) - \sum_{t=1}^{T} \langle \eta_t h_t, \Delta_{t+1}\rangle$$

We can bound the norm of Δ_t by considering that (a) it start at 0 and (b) at each time step it increases by at most ∇_t and is multiplied by $1 - \gamma^2$. This implies that $\|\Delta_t\| \leq cG$ where $c = \frac{\sqrt{1-\gamma^2}}{1-\sqrt{1-\gamma^2}} < \frac{2}{\gamma^2}$.

From here we can get a final bound:

$$\sum_{t=1}^{T} \mathcal{R}[f^*] \geq \sum_{t=1}^{T} \mathcal{R}[f_t] - \frac{c^2G^2}{m}(1 + \ln T) - \frac{2c^2G^2}{mT} - \frac{c^2G^2}{m}(1 + \ln T)$$

■

Theorem 2.5.4. *Let $\mathcal{R}$ be a convex functional over $\mathcal{F}$. Let $\mathcal{H} \subset \mathcal{F}$ be a restriction set with edge γ for every Δ that is projected on to $\mathcal{H}$. Let $\|\nabla\mathcal{R}[f]\| \leq G$ and $\|f\| \leq F$ for all $f \in \mathcal{F}$. Let $f^* = \arg\min_{f\in\mathcal{F}}\mathcal{R}[f]$. Let $c = \frac{2}{\gamma^2}$. Given a starting point f_0 and step size $\eta_t = \frac{1}{\sqrt{t}}$, after T iterations of Algorithm 2.4 we have:*

$$\frac{1}{T}\sum_{t=1}^{T}[\mathcal{R}[f_t] - \mathcal{R}[f^*]] \leq \frac{F^2}{2\sqrt{T}} + \frac{c^2G^2}{\sqrt{T}} + \frac{c^2G^2}{2T^{\frac{3}{2}}}.$$

Proof. Similar to the last few proofs, we get a result similar to the standard gradient version, with the error term from the last proof:

$$\sum_{t=1}^{T} \mathcal{R}[f^*] \geq \sum_{t=1}^{T} \mathcal{R}[f_t] + \sum_{t=1}^{T} \frac{1}{2}\|f_t - f^*\|^2(\frac{1}{\eta_t} - \frac{1}{\eta_{t+1}}) - \sum_{t=1}^{T} \frac{\eta_{t+1}}{2}\|h_t\|^2 - (\langle f^* - f_{T+1}, \Delta_{t+1}\rangle - \frac{\sqrt{T}}{2}\|f^* - f_{T+1}\|^2) - \sum_{t=1}^{T} \langle \eta_{t+1} h_t, \Delta_{t+1}\rangle$$

Using the bound on $\|\Delta_t\| \leq c$ from above and setting $\eta_t = \frac{1}{\sqrt{t}}$:

$$\sum_{t=1}^{T} \mathcal{R}[f^*] \geq \sum_{t=1}^{T} \mathcal{R}[f_t] - \frac{F^2\sqrt{T}}{2} - c^2G^2\sqrt{T} - \frac{c^2G^2}{2\sqrt{T}}$$

giving the final bound. ■

Again, the results are similar bounds to those from the non-restricted case. Like the previous proof, the extra terms in the bound come from the penalty paid in projection errors at each time step, but here the residual serves as a mechanism for pushing the error back to later projections. The analysis relies on a bound on the norm of the residual Δ, derived by observing that it is increased by at most the norm of the gradient and then multiplicatively decreased in projection due to the edge requirement. This bound on the size of the residual presents itself in the c term present in the bound.

In terms of efficiency, these two algorithms are similarly matched. For the strongly convex case, the repeated projection algorithm uses $O(T^2)$ weak learners to obtain an average regret of $O(\frac{\ln T}{T} + \frac{1}{\gamma^2 T})$, while the residual algorithm uses $O(T)$ weak learners and has average regret $O(\frac{\ln T}{\gamma^4 T})$. The major difference lies in frequency of the gradient evaluation, where the repeated projection algorithm evaluates the gradient much less often than the than the residual algorithm.

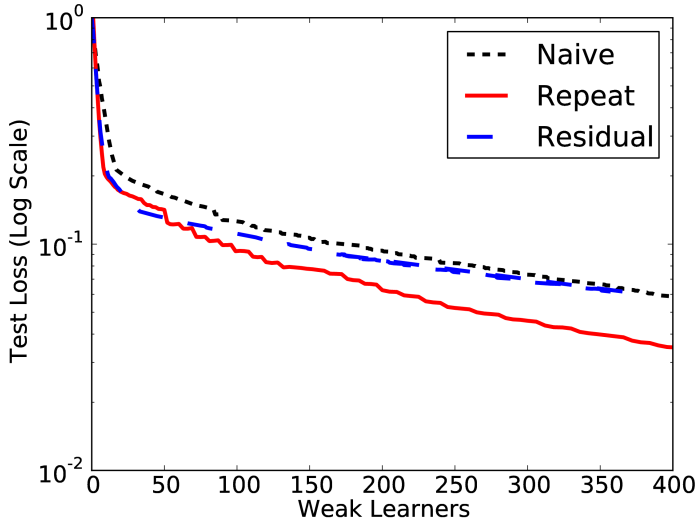


Figure 2.3: Test set loss vs number of weak learners used for a maximum margin structured imitation learning problem for all three restricted gradient algorithms. The algorithms shown are the naive use of the basic projection (black dashed line), repeated projection steps (red solid line), and the residual projection algorithm (blue long dashed line).

2.6 Experiments

We now present experimental results for these new algorithms on three tasks: an imitation learning problem, a ranking problem and a set of sample classification tasks.

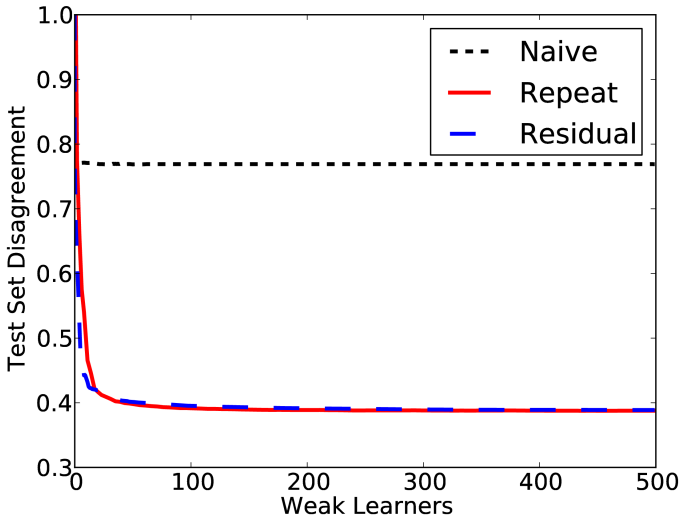


Figure 2.4: Test set disagreement (fraction of violated constraints) vs number of weak learners used for the MSLR-WEB10K ranking dataset for all three restricted gradient algorithms. The algorithms shown are the naive use of the basic projection (black dashed line), repeated projection steps (red solid line), and the residual projection algorithm (blue long dashed line).

The first experimental setup is an optimization problem which results from the Maximum Margin Planning [Ratliff et al., 2009] approach to imitation learning. In this setting, a demonstrated policy is provided as example behavior and the goal is to learn a cost function over features of the environment which produce policies with similar behavior.

Previous attempts in the literature have been made to adapt boosting to this setting [Ratliff et al., 2009, Bradley, 2009], similar to the naive algorithm presented here, but no convergence results for this settings are known.

Figure 2.3 shows the results of naively applying the basic projected gradient algorithm, as well as running the two new algorithms presented here on a sample planning dataset from this domain. The weak learners used were neural networks with 5 hidden units each.

The second experimental setting is a ranking task from the Microsoft Learning to Rank Datasets, specifically MSLR-WEB10K [Microsoft, 2010], using the ranking version of the hinge loss and decision stumps as weak learners. Figure 2.4 shows the test set disagreement (the percentage of violated ranking constraints) plotted against the number of weak learners.

As a final test, we ran our boosting algorithms on several multiclass classification tasks from the UCI Machine Learning Repository [Frank and Asuncion, 2010], using the ‘connect4’, ‘letter’, ‘pendigits’ and ‘satimage’ datasets. All experiments used the multiclass extension to the hinge loss [Crammer and Singer, 2002], along with multiclass decision stumps for the weak learners. Results are given in Figure 2.5.

Of particular interest are the experiments where the naive approach to restricted gradient de-

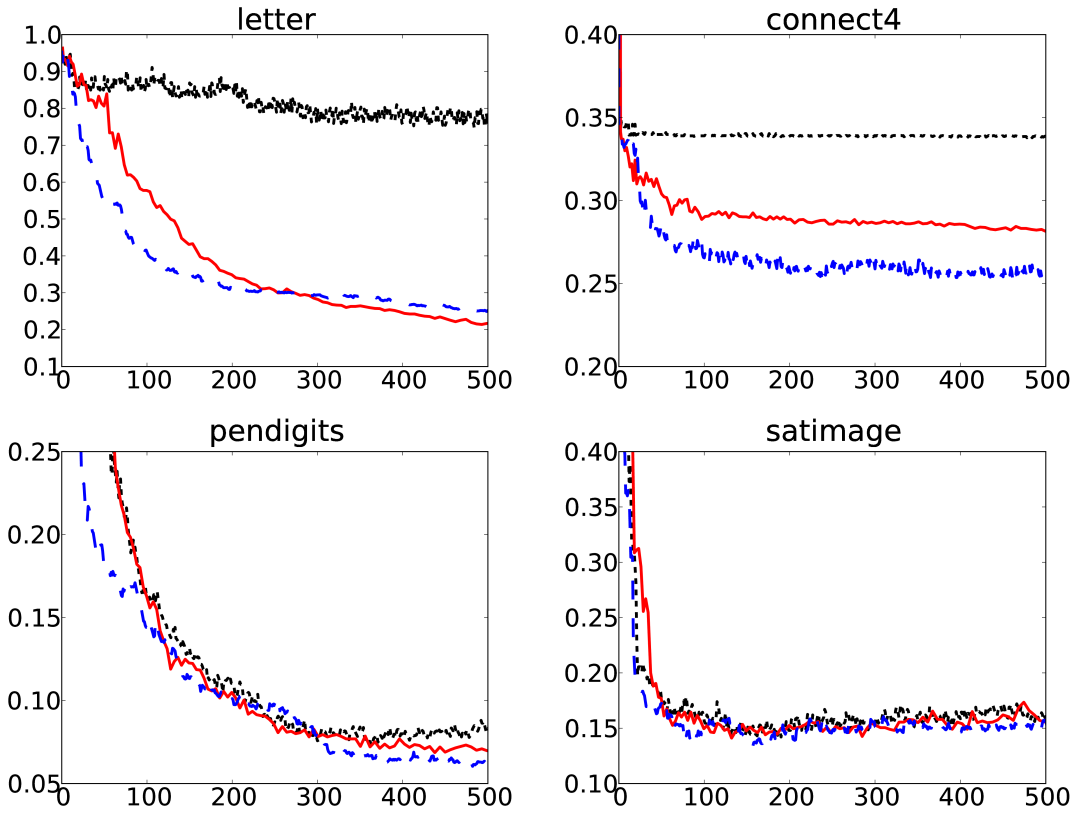


Figure 2.5: Test set classification error on multiclass classification experiments over the UCI 'connect4', 'letter', 'pendigits' and 'satimage' datasets. The algorithms shown are the naive use of the basic projection (black dashed line), repeated projection steps (red solid line), and the residual projection algorithm (blue long dashed line).

scent clearly fails to converge ('connect4' and 'letter'). In line with the presented convergence results, both non-smooth algorithms approach optimal training performance at relatively similar rates, while the naive approach cannot overcome the particular conditions of these datasets and fails to achieve strong performance. In these cases, the naive approach repeatedly cycles through the same weak learners, impeding further optimization progress.

Chapter 3

Functional Gradient Extensions

In this chapter we detail two extensions to the functional gradient techniques described in Chapter 2. The first extension covers the structured prediction setting, where each example has a corresponding *structured output* we wish to generate, consisting of a number of individual predictions over the relevant structural features of the problem. For example, the structured output might be a label for every word in a sentence, or every pixel in an image. Most notably different from the previous chapter, in this domain we will learn boosted learners that rely on the values of previous predictions at each iteration of boosting, so the learned function can account between relationships between structurally related predictions.

This change introduces a unique type of overfitting which often results in a cascade of failures in practice, due to the reliance on potentially overfit previous predictions at training time. To address this, we introduce a second extension which is a *stacked* version of functional gradient methods. This algorithm improves robustness to the overfitting that occurs when predictions from the early weak learners in are reused as input features to later weak learners.

3.1 Structured Boosting

3.1.1 Background

In the structured prediction setting, we are given inputs $x \in \mathcal{X}$ and associated structured outputs $y \in \mathcal{Y}$. The goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that minimizes some risk $\mathcal{R}[f]$, typically evaluated pointwise over the inputs:

$$\mathcal{R}[f] = \mathbb{E}_{\mathcal{X}}[\ell(f(x))], \tag{3.1}$$

similar to the pointwise loss discussed in the previous chapter in Equation (2.4).

We will further assume that each input and output pair has some underlying structure, such as the graph structure of graphical models, that can be utilized to predict portions of the output locally. Let j index these structural elements. We then assume that a final structured output y can

be represented as a variable length vector $(y_1, \ldots, y_J)$, where each element y_j lies in some vector space $y_j \in \mathcal{Y}'$. For example, these outputs could be the probability distribution over class labels for each pixel in an image, or distributions of part-of-speech labels for each word in a sentence. Similarly we can compute some features x_j representing the portion of the input which corresponds to a given output, such as features computed over a neighborhood around a pixel in an input image.

As another example, consider labeling tasks such as part-of-speech tagging. In this domain, we are given a set of input sentences $\mathcal{X}$, and for each word j in a given sentence, we want to output a vector $\hat{y}_j \in \mathbb{R}^K$ containing the scores with respect to each of the K possible part-of-speech labels for that word. This sequence of vectors for each word is the complete structured prediction $\hat{y}$. An example loss function for this domain would be the multiclass log-loss, averaged over words, with respect to the ground truth parts-of-speech.

Along with the encoding of the problem, we also assume that the structure can be used to reduce the scope of the prediction problem, as in graphical models. One common approach to generating predictions on these structures is to use a policy-based or iterative decoding approach [Cohen and Carvalho, 2005, Daume III et al., 2009, Tu and Bai, 2010, Socher et al., 2011, Ross et al., 2011], instead of probabilistic inference over a graphical model. In order to model the contextual relationships among the outputs, these iterative approaches commonly perform a sequence of predictions, where each update relies on previous predictions made across the structure of the problem.

Let $N(j)$ represent the locally connected elements of j, such as the locally connected factors of a node j in a typical graphical model. For a given node j, the predictions over the neighboring nodes $\hat{y}_{N(j)}$ can then be used to update the prediction for that node. For example, in the character recognition task, the predictions for neighboring characters can influence the prediction for a given character, and be used to update and improve the accuracy of that prediction.

In the iterative decoding approach a predictor ϕ is iteratively used to update different elements $\hat{y}_j$ of the final structured output:

$$\hat{y}_j = \phi(x_j, \hat{y}_{N(j)}), \tag{3.2}$$

using both the features x_j of that element and current predictions $\hat{y}_{N(j)}$ of the neighboring elements. In the message passing analogy, these current predictions are the messages that are passed between nodes, and used for updating the current prediction at that node.

A complete policy then consists of a strategy for selecting which elements of the structured output to update, coupled with the predictor for updating the given outputs. Typical approaches include randomly selecting elements to update, iterating over the structure in a fixed ordering, or simultaneously updating all predictions at all iterations. As shown by Ross et al. [2011], this iterative decoding approach can is equivalent to message passing approaches used to perform inference over graphical models, where each update encodes a single set of messages passed to one node in the graphical model. For example, the message passing behavior of Loopy Belief Propagation [Pearl, 1988] can be described by this iterative decoding approach [Ross et al., 2011].

3.1.2 Weak Structured Predictors

We now adapt the functional gradient methods discussed in Chapter 2 to the structured prediction setting by detailing an additive structured predictor. To accomplish this, we will adapt the policy-based iterative decoding approach discussed in Section 3.1.1 to use an additive policy instead of one which replaces previous predictions.

In the iterative decoding described previously, recall that we have two components, one for selecting which elements to update, and another for updating the predictions of the given elements. Let $\mathcal{S}^t$ be the set of components selected for updating at iteration t. For current predictions y^t we can re-write the policy for computing the predictions at the next iteration of the iterative decoding procedure as:

$$\hat{y}_j^{t+1} = \begin{cases} \phi(x_j, \hat{y}^t_{N(j)}) & \text{if } j \in \mathcal{S}^t \\ \hat{y}_j^t & \text{otherwise} \end{cases}. \tag{3.3}$$

The additive version of this policy instead uses weak predictors h, each of which maps both the input data and previous structured output to a more refined structured output, $h : \mathcal{X} \times \mathcal{Y} \to \mathcal{Y}$:

$$\hat{y}^{t+1} = \hat{y}^t + h(x, \hat{y}^t). \tag{3.4}$$

Note that, unlike in Chapter 2, we are now augmenting each weak learner h to also take as input the current prediction, $\hat{y}$.

We can build a weak predictor h which performs the same actions as the previous replacement policy by considering weak predictors with two parts: a function h_{S} which selects which structural elements to update, and a predictor h_{P} which runs on the selected elements and updates the respective pieces of the structured output.

The selection function h_{S} takes in an input x and previous prediction $\hat{y}$ and outputs a set of structural nodes $\mathcal{S} = \{j_1, j_2, \ldots\}$ to update. For each structural element selected by h_{S}, the predictor h_{P} takes the place of ϕ in the previous policy, taking $(x_j, \hat{y}_{N(j)})$ and computing an update for the prediction $\hat{y}_j$.

Returning to the part-of-speech tagging example, possible selection functions would select different chunks of the sentence, either individual words or multi-word phrases using some selection criteria. Given the set of selected elements, a prediction function would take each selected word or phrase and update the predicted distribution over the part-of-speech labels using the features for that word or phrase.

Using these elements we can write the weak predictor h, which produces a structured output $(h(\cdot)_1, \ldots, h(\cdot)_J)$, as

$$h(x, \hat{y}^t)_j = \begin{cases} h_{\text{P}}(x_j, \hat{y}^t_{N(j)}) & \text{if } j \in h_{\text{S}}(x, \hat{y}) \\ 0 & \text{otherwise} \end{cases}, \tag{3.5}$$

or alternatively we can write this using an indicator function:

$$h(x, \hat{y}^t)_j = \mathbb{1}(j \in h_{\text{S}}(x, \hat{y}^t)) h_{\text{P}}(x_j, \hat{y}^t_{N(j)}). \tag{3.6}$$

3.1.3 Functional Gradient Projection

In order to use the functional gradient framework discussed in the previous chapter, we need to be able to complete the projection operation over the $\mathcal{H}$ given in Equations (2.11-2.12). We assume that we are given a fixed set of possible selection functions, $\mathcal{H}_{\text{S}}$, and a set of L learning algorithms, $\{\mathcal{A}_l\}_{l=1}^{L}$, where $\mathcal{A} : \mathcal{D} \rightarrow \mathcal{H}_{\text{P}}$ generates a predictor given a training set D.

In practice, these algorithms may be different methods for generating regressors, classifiers, or other weak learners tailored to the specific problem. The reason for this distinction between the selection and prediction functions is that, in practice, the selection functions are often problem specific and cannot be trained to target a given gradient, like weak learners often are. Instead, we will use an enumeration strategy to find the best selection function, and train the prediction functions to target a given functional gradient.

Consider the loss function given in Equation (3.1). This function is actually a function of the vector of outputs $(\hat{y}_1, \ldots)$. Recall from the previous chapter and Equation (2.7) that the gradient ∇ at each input x will simply be the gradient of the loss ℓ at the current output,

$$\nabla(x) = \nabla_{f(x)} \ell(f(x)).$$

In the structured prediction setting, we are actually concerned with each individual component of the gradient $\nabla(x)_j$, corresponding to output y_j. This gradient component is simply

$$\nabla(x)_j = \frac{\partial \ell(f(x))}{\partial f(x)_j}, \tag{3.7}$$

or, the gradient of the loss with respect to the partial structured prediction $\hat{y}_j = f(x)_j$.

Given a fixed selection function h_{S} and current predictions $\hat{y}$, we can build a dataset appropriate for training weak predictors h_{P} as follows. In order to minimize the projection error in Equation (2.12) for a predictor h of the form in Equation (3.6), we only need to find the prediction function h_{P} that minimizes

$$h_{\text{P}}^{*} = \underset{h_{\text{P}} \in \mathcal{H}_{\text{P}}}{\arg\min}\, \mathbb{E}_{\mathcal{X}} \left[\sum_{j \in h_{\text{S}}(x, \hat{y})} \left\| \nabla(x)_j - h_{\text{P}}(x_j, \hat{y}_{N(j)}) \right\|^2 \right]. \tag{3.8}$$

This optimization problem is equivalent to minimizing weighted least squares error over the dataset

$$\begin{aligned} D &= \bigcup_{x} \bigcup_{j \in h_{\text{S}}(x, \hat{y})} \{(\psi_j, \nabla(x)_j)\}, \\ &= \text{gradient}(f, h_{\text{S}}), \end{aligned} \tag{3.9}$$

where $\psi_j = \psi(x_j, \hat{y}_{N(j)})$ is a feature descriptor for the given structural node, and $\nabla(x)_j$ is its target. In order to model contextual information, ψ is drawn from both the raw features x_j for the given element and the previous locally neighboring predictions $\hat{y}_{N(j)}$.

Now, we can use this modified weak learner and projection operation in the functional gradient descent framework from Chapter 2. In order to complete gradient projection operation over all weak learners in the set defined by Equation (3.6), we simply enumerate all selection strategies h_{S}. Then, for each selection strategy and each learning algorithm $\{\mathcal{A}_l\}_{l=1}^{L}$, we can use Equation (3.8), via the dataset in Equation (3.9) to generate a candidate weak prediction function h_{P} for that selector, algorithm pair. The pair $h_{\mathrm{S}}, h_{\mathrm{P}}$ can be used to define a structured weak learner h as in Equation (3.6), and the best overall weak learner can be selected as the projected gradient.

Algorithm 3.1 Structured Functional Gradient Descent

Given: objective $\mathcal{R}$, set of selection functions $\mathcal{H}_{\mathrm{S}}$, set of L learning algorithms $\{\mathcal{A}_l\}_{l=1}^{L}$, number of iterations T, initial function f_0.
for $t = 1, \ldots, T$ **do**
  $\mathcal{H}^* = \emptyset$
  for $h_{\mathrm{S}} \in \mathcal{H}_{\mathrm{S}}$ **do**
    Create dataset $D = \text{gradient}(f_{t-1}, h_{\mathrm{S}})$ using Equation (3.9).
    for $\mathcal{A} \in \{\mathcal{A}_1, \ldots, \mathcal{A}_L\}$ **do**
      Train $h_{\mathrm{P}} = \mathcal{A}(D)$
      Define h from h_{S} and h_{P} using Equation (3.6).
      $\mathcal{H}^* = \mathcal{H}^* \cup \{h\}$
    end for
  end for
  Let $\nabla(x) = \nabla_{f(x)}\ell(f(x))$ for all x.
  $h_t = \arg\min_{h \in \mathcal{H}^*} \mathbb{E}_x[\|\nabla(x) - h(x)\|^2]$
  Select a step size α_t.
  $f_t = f_{t-1} + \alpha_t h_t$
end for

Algorithm 3.1 summarizes the structured version of functional gradient descent. It enumerates the candidate selection functions, h_{S}, creates the training dataset defined by Equation (3.9), and then generates a candidate prediction function h_{P} using each weak learning algorithm.

We will make use of this algorithm in Chapter 7, when we examine an anytime structured prediction approach. For more details on applications of this structured functional gradient approach and practical implementation concerns, see Chapter 7.

3.2 Stacked Boosting

When training models that incorporate previous predictions, such as the structured prediction approach discussed in Section 3.1, the risk of overfitting is typically of large concern. In this section, we examine the use of *stacking*, a method for training multiple simultaneous predictors in order to

simulate the overfitting in early predictions, and show how to use this approach to reduce overfitting in functional gradient methods which re-use previous predictions.

Originally from a different domain, the concept of stacking Wolpert [1992], Cohen and Carvalho [2005] is an approach for reducing the overfitting in a model due to the re-use of previous predictions. Essentially this method trains multiple copies of a given predictor while holding out portions of the dataset, in a manner similar to cross-validation. Each predictor is then run on the held-out data to generate "unbiased" predictions for use as inputs to later predictors, mitigating the impact of overfitting on those predictions. This approach has proven to be useful in structured prediction settings Cohen and Carvalho [2005], Munoz et al. [2010] such as computer vision, where it is common to build sequential predictors which use neighboring and previous predictions as contextual information to improve overall performance.

It is this stacking approach which we will now examine and extend to the functional gradient setting.

3.2.1 Background

The stacking method is originally a method for training feed-forward networks, or sequences of predictors which re-use previous predictions as inputs at each point in the sequence. In stacked forward training, a set of predictors is trained using a sequential approach that trains each successive predictor iteratively using the outputs from previously trained ones. This approach is common in structured prediction tasks such as vision where iterated predictions are used allow for smoothing of neighboring regions or when the structure of lower level features is selected apriori and trained independently of later, more complex feature representations.

Assume we are given a dataset $\mathcal{D}^0$ of examples and labels $\{(x_n, y_n)\}_{n=0}^N$. We model the feed-forward ensemble of K learners as a sequence of predictors $f^1, \ldots, f^K$, with the output of predictor k given as

$$x_n^k = f^k(x_n^{k-1}),$$

with the initial input $x_n^0 = x_n$.

Assume that for each iteration k, there is a learning algorithm $\mathcal{A}^k(D)$ for generating the predictor f^k for that iteration, using predictions from the previous iterations x_n^{k-1} and labels y_n. That is, having trained the previous functions $f^1, \ldots, f^{k-1}$, the next predictor is trained by building a dataset

$$\mathcal{S}^k = \{(x_n^{k-1}, y_n\}_{n=0}^N,$$

and then training the current layer

$$f^k = \mathcal{A}^k(\mathcal{S}^k). \tag{3.10}$$

This method is not robust to overfitting, however, as errors in early predictors are re-used for training later predictors, while unseen test data will likely generate less accurate predictions or low level features. If early predictors in the sequence overfit to the training set, later predictors will be

trained to rely on these overfit inputs, potentially overfitting further to the training set and leading to a cascade of failures.

The stacking method Cohen and Carvalho [2005] is a method for reducing the overall generalization error of a sequence of trained predictors, by attempting to generate an unbiased set of previous predictions for use in training each successive predictor. This is done by training multiple copies of each predictor on different portions of the data, in a manner similar to cross-validation, and using these copies to predict on unseen parts of the data set.

More formally, we split the dataset $\mathcal{S}^k$ in to J equal portions $\mathcal{S}^k_1, \ldots, \mathcal{S}^k_J$, and for each predictor f^k train an additional J copies f^k_j. Each copy is trained on the dataset *excluding* the corresponding fold, as in J-fold cross-validation:

$$f^k_j = \mathcal{A}^k(\mathcal{S}^k \setminus \mathcal{S}^k_j). \tag{3.11}$$

Each of the copies is then used to generate predictions on the held-out portion of the data which are used to continue the training by building a dataset of the held-out predictions:

$$\mathcal{S}^{k+1} = \cup_{j=1}^{J} \left\{ (f^k_j(x), y) \mid (x, y) \in \mathcal{S}^k_j \right\}. \tag{3.12}$$

The predictor f_k for the final sequence is still trained on the whole dataset $\mathcal{S}^k$, as in (3.10) and returned in the final model. The stacked copies are only used to generate the predictions for training the rest of the sequence, and are then discarded.

A complete description of stacked forward training is given in Algorithm 3.2.

Algorithm 3.2 Stacked Forward Training

Given: initial dataset $\mathcal{S}^0$, training algorithms $\mathcal{A}^k$, number of stacking folds J.
for $k = 1, \ldots, K$ **do**
  Let $f^k = \mathcal{A}^k(\mathcal{S}^k)$.
  Split $\mathcal{S}^k$ into equal parts $\mathcal{S}^k_1, \ldots, \mathcal{S}^k_J$.
  For $j = 1, \ldots, J$ let $f^k_j = \mathcal{A}^k(\mathcal{S} \setminus \mathcal{S}^k_j)$.
  Let $\mathcal{S}^{k+1} = \cup_{j=1}^{J} \left\{ (f^k_j(x), y) \mid (x, y) \in \mathcal{S}^k_j \right\}$.
end for
return $\left(f^1, \ldots, f^K\right)$.

3.2.2 Stacked Functional Gradient Methods

Now we want to adapt this stacking method to the domain of functional gradient methods. One key difference between the stacking approach used in Section 3.2.1 and the functional gradient approach is that stacking was originally developed for networks of predictors which use *only* previous predictions as inputs at each layer, while in the functional gradient method, we still want to retain the example x as an input in addition to previous predictions.

Algorithm 3.3 Stacked Functional Gradient Descent

Given: starting point f_0, step size schedule $\{\eta_t\}_{t=1}^T$, number of stacking folds K.
Split training data $\mathcal{X}$ into equal parts $\mathcal{X}_1, \ldots, \mathcal{X}_K$.
Let $f_{k,0} = f_0$.
for $t = 1, \ldots, T$ **do**
  for $k = 1, \ldots, K$ **do**
    Let $\mathcal{X}_k^{\complement} = \mathcal{X} \setminus \mathcal{X}_k$.
    Let $\hat{\mathcal{Y}}_k^{\complement} = \{f_{k,t-1}(x) \mid x \in \mathcal{X}_k^{\complement}\}$.
    Let $\hat{\mathcal{Y}}_k = \{f_{k,t-1}(x) \mid x \in \mathcal{X}_k\}$.
    Compute a subgradient $\nabla_{k,t} \in \partial\mathcal{R}[f_{k,t-1}]$ over only points in $\mathcal{X}_k^{\complement}$.
    Compute $h_k^* = \text{Proj}\,(\nabla_{k,t}, \mathcal{H})$, again only over points in $\mathcal{X}_k^{\complement}$ and using $\hat{\mathcal{Y}}_k^{\complement}$ as previous predictions.
    Update f_k: $f_{k,t} = f_{k,t-1} - \eta_t h_k^*$.
  end for
  Let $\hat{\mathcal{Y}} = \bigcup_k \hat{\mathcal{Y}}_k$.
  Compute a subgradient $\nabla_t \in \partial\mathcal{R}[f_{t-1}]$ over all points in $\mathcal{X}$.
  Compute $h^* = \text{Proj}\,(\nabla, \mathcal{H})$ over all points in $\mathcal{X}$ and using $\hat{\mathcal{Y}}$ as previous predictions.
  Update f: $f_t = f_{t-1} - \eta_t h^*$.
end for

Consider the following weak learner h which takes both example and previous predictions as inputs:

$$h(x, \hat{y}) : \mathcal{X} \times \mathcal{Y} \to \mathcal{Y}.$$

One example is the structured weak learner discussed in Section 3.1, and given in Equation 3.6.

We can define the final output of a boosted ensemble of such weak learners as

$$f(x) = \sum_t h_t(x, \hat{y}_t),$$

where $\hat{y}_t$ is given as the prediction up to weak learner t:

$$\hat{y}_t = \sum_{i=1}^{t} h_t(x, \hat{y}_i).$$

We want to use the stacking method to generate held-out version of the predictions for use as input when computing new weak learners. To do this, we will follow the same general procedure as outlined above for feed-forward stacking.

We will maintain the real boosted predictor f, along with copies f_k for each of K folds of the training data. At training time, each copy f_k is trained on all data *except* fold k, and using its own predictions as previous predictions. The true predictor f is trained using all the data, but

the previous predictions are drawn from the outputs of each copy f_k, run on its respective fold of the data $\mathcal{X}_k$ which it was not previously trained on. At test time, only the predictor f which was trained on all data will be used for prediction.

Algorithm 3.3 gives the stacked version of functional gradient descent. This method can be combined with other functional gradient methods fairly easily, such as the structured functional gradient approach detailed earlier, by simply following the same strategy of maintaining K copies of boosted predictor and using each copy to compute held-out predictions. Later, in Chapter 7, we will combined both of these approaches to build a stacked, structured functional gradient learner.

Part II

Greedy Optimization

Chapter 4

Budgeted Submodular Function Maximization

In this chapter we analyze the performance of greedy and approximately greedy algorithms for budgeted, approximate submodular maximization. We will be using a version of approximate submodularity which includes both a multiplicative and additive relaxation from the standard definition of submodularity. For this setting, we show that greedy approaches achieve approximation bounds with respect to a subset of all arbitrary budgets corresponding to the costs of each successive subsequence selected. Finally, we show that, if an approximation bound is desired for any arbitrary budget, a modification of the greedy algorithm can achieve a bi-criteria approximation in both value and time for arbitrary budgets.

4.1 Background

In this chapter we will be analyzing approaches for maximizing positive set functions $F : 2^{\mathcal{X}} \rightarrow \mathbb{R}$, $F(S) > 0$ over elements $\mathcal{X}$, where $2^{\mathcal{X}}$ is the power set of $\mathcal{X}$. We will be building off of a large body of work focusing on *submodular* functions. A function F is submodular if, for all $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{X}$

$$F(\{x\} \cup \mathcal{A}) - F(\mathcal{A}) \geq F(\{x\} \cup \mathcal{B}) - F(\mathcal{B}).$$

An equivalent definition, which we will build off of later relates the gain in the value of F when adding a whole set to the gain when adding each element individually. A function F is submodular for all $\mathcal{S}, \mathcal{A} \subset \mathcal{X}$

$$F(\mathcal{S} \cup \mathcal{A}) - F(\mathcal{A}) \leq \sum_{x \in \mathcal{S}} F(\mathcal{A} \cup \{x\}) - F(\mathcal{A}). \tag{4.1}$$

We will further restrict our analysis to monotone submodular functions, that is, functions F such that $F(\mathcal{A}) \leq F(\mathcal{B})$ if $\mathcal{A} \subseteq \mathcal{B}$.

In the budgeted setting, every element in $\mathcal{X}$ is associated with a positive cost $c : \mathcal{X} \rightarrow \mathbb{R}$, $c(x) > 0$. The cost of a set of elements $\mathcal{S}$ is the modular function $c(\mathcal{S}) = \sum_{x \in \mathcal{S}} c(x)$.

The budgeted monotone submodular maximization problem is then to maximize a set function F subject to a constraint B on the total cost of the set:

$$\begin{aligned} &\arg\max_{\mathcal{S}} F(\mathcal{S}) \\ &c(\mathcal{S}) \leq B. \end{aligned} \tag{4.2}$$

When performing our analysis, we will be working both with sequences of elements, e.g. the sequence of selection made by a given algorithm, and sets of elements corresponding to particular points in said sequences. Given a sequence $S = s_1, \ldots$ and a given budget C, we can define the resulting set at that budget to be $\mathcal{S}_{\langle C \rangle} = \{s_1, \ldots, s_k\}$ such that $\sum_{i=1}^{k} c(s_i) \leq C$. Similarly, define the set $\mathcal{S}_k$ to be $\{s_1, \ldots, s_k\}$, and $\mathcal{S}_0 = \emptyset$.

As discussed in the discussion of related work in Section 1.3, previous work [Khuller et al., 1999, Krause and Guestrin, 2005, Leskovec et al., 2007, Lin and Bilmes, 2010] has included a number of algorithms and corresponding approximation bounds for this setting. These approaches range from variations on the cost-greedy algorithm to much more complex strategies, and have approximation bounds with factors of $\frac{1}{2}(1-\frac{1}{e})$ and $(1-\frac{1}{e})$, extending the original result of Nemhauser et al. [1978] for the unit-cost, or cardinality constrained case.

The key difference between our analysis here and these previous results is that these results all require that the budget be known apriori. For example, one of the results of Krause and Guestrin [2005] which achieves a $\frac{1}{2}(1 - \frac{1}{e})$ approximation uses a modified greedy algorithm which selects either the result of the cost-greedy algorithm, or the single largest element with cost less than the budget.

Unfortunately, for our purposes we want a single, budget-agnostic algorithm which produces a sequence of elements with good performance at any budget. Approaches such as the previous example both target a fixed budget and do not produce a single sequence for all budgets. If the budget is increased, the selected set may change completely, whereas we want a method such that increasing the budget only adds elements to the currently selected set.

As we will discuss in Section 4.4, in general it is impossible to have a budget-agnostic algorithm which achieves approximation bounds for all budgets, but a small tweak to the cost-greedy algorithm does produce a sequence which achieves a *bi-criteria approximation*, which approximates the optimal set in both value and in cost.

4.2 Approximate Submodularity

Unliked the submodular functions which we discussed in Section 4.1, we want to analyze the performance of greedy algorithms on functions which behave like submodular functions to some degree, but are not strictly submodular. Building off of the requirement given in Equation (4.1), Das and Kempe [2011] give a definition of approximate submodularity which uses a multiplicative ratio $\gamma \in [0, 1]$ which they call the *submodularity ratio*. In this work, we extend this definition of

approximate submodularity to also allow for an additive error term δ, similar to the approximate submodularity definition utilized by Krause and Cehver [2010].

Definition 4.2.1 (Approximate Submodularity)**.** *A function F is (γ, δ)-approximately submodular if:*

$$\gamma \left[F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})\right] - \delta \leq \sum_{x \in \mathcal{S}} \left[F(\mathcal{A} \cup \{x\}) - F(\mathcal{A})\right],$$

for all $\mathcal{S}, \mathcal{A} \subseteq \mathcal{X}$.

As expected, this notion of approximate submodularity also extends the traditional definition of submodularity given in Equation (4.1), with any submodular function being $(1, 0)$-approximately submodular. Further, for $\delta = 0$ this definition reduces to the one given by Das and Kempe [2011].

4.2.1 Greedy Algorithm Analysis

Algorithm 4.1 Greedy Algorithm

Given: objective function F, elements $\mathcal{X}$
Let $\mathcal{G}_0 = \emptyset$.
for $j = 1, \ldots$ **do**
  Let $g_j = \arg\max_{x \in \mathcal{X}} \frac{F(\mathcal{G}_{j-1} \cup \{x\}) - F(\mathcal{G}_{j-1})}{c(x)}$.
  Let $\mathcal{G}_j = \mathcal{G}_{j-1} \cup \{g_j\}$.
end for

We will now analyze the standard greedy algorithm (given in Algorithm 4.1) for the budgeted submodular maximization problem, operating on a set function F that is approximately submodular according to Definition 4.2.1.

As shown in Algorithm 4.1, the cost-greedy algorithm iteratively selects a sequence $G = (g_1, \ldots)$ using:

$$g_j = \arg\max_{x \in \mathcal{X}} \left[\frac{F(\mathcal{G}_{j-1} \cup \{x\}) - F(\mathcal{G}_{j-1})}{c(x)}\right]. \tag{4.3}$$

We now present a bound that shows that the cost-greedy algorithm is nearly optimal for approximately submodular functions. The analysis is a combination of the cost-based greedy analysis of Streeter and Golovin [2008], generalized to handle the approximate submodular case as in Das and Kempe [2011]. Similar to Krause and Golovin [2012], we also handle the case where the greedy list and optimal list are selected using different budgets.

First, we need to adapt the approximate submodularity definition given in Definition 4.2.1 to a bound that also relates the costs of the elements and combined set.

Lemma 4.2.2. *If a function F is γ, δ-approximately submodular then for any $\mathcal{A}, \mathcal{S} \subseteq \mathcal{V}$:*

$$\frac{\gamma\left[F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})\right] - \delta}{c(\mathcal{S})} \leq \max_{x \in \mathcal{S}} \left[\frac{F(\mathcal{A} \cup \{x\}) - F(\mathcal{A})}{c(x)}\right].$$

Proof. By Definition 4.2.1, we have:

$$\begin{aligned}
\gamma\left[F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})\right] - \delta &\leq \sum_{x \in \mathcal{S}} \left[F(\mathcal{A} \cup \{x\}) - F(\mathcal{A})\right] \\
&\leq \sum_{x \in \mathcal{S}} \max_{x' \in \mathcal{S}} \left[\frac{F(\mathcal{A} \cup \{x'\}) - F(\mathcal{A})}{c(x')}\right] c(x) \\
&\leq \left(\max_{x \in \mathcal{S}} \left[\frac{F(\mathcal{A} \cup \{x\}) - F(\mathcal{A})}{c(x)}\right]\right) \left(\sum_{x \in \mathcal{S}} c(x)\right).
\end{aligned}$$

Dividing boths sides by $c(\mathcal{S}) = \sum_{x \in \mathcal{S}} c(x)$ completes the proof. ■

Now, we can use this result to bound the gap between the optimal set and the set selected by the greedy algorithm at each iteration.

Lemma 4.2.3. *Let s_j be the value of the maximum in Equation (4.3) evaluated by the greedy algorithm at iteration j. Then for all sequences S and total costs C:*

$$F(\mathcal{S}_{\langle C \rangle}) \leq F(\mathcal{G}_{j-1}) + \frac{C s_j + \delta}{\gamma}.$$

Proof. By Lemma 4.2.2 we have:

$$\begin{aligned}
\frac{\gamma\left[F(\mathcal{G}_{j-1} \cup \mathcal{S}_{\langle C \rangle}) - F(\mathcal{G}_{j-1})\right] - \delta}{c(\mathcal{S}_{\langle C \rangle})} &\leq \max_{x \in \mathcal{S}_{\langle C \rangle}} \left[\frac{F(\mathcal{G}_{j-1} \cup \{x\}) - F(\mathcal{G}_{j-1})}{c(x)}\right] \\
&\leq s_j
\end{aligned}$$

By monotonicity we have $F(\mathcal{S}_{\langle C \rangle}) \leq F(\mathcal{G}_{j-1} \cup \mathcal{S}_{\langle C \rangle})$, and by definition $c(\mathcal{S}_{\langle C \rangle}) \leq C$ giving:

$$F(\mathcal{S}_{\langle C \rangle}) \leq F(\mathcal{G}_{j-1}) + \frac{C s_j + \delta}{\gamma}.$$

■

Now, using Lemma 4.2.3, we can derive the actual approximation bound for the greedy algorithm. Like previous results [Das and Kempe, 2011], for a (γ, δ)-approximately submodular function, this bound includes the multiplicative term γ in the multplicative approximation term, but has an additional additive term dependent on the additive term δ.

Theorem 4.2.4. *Let $G = (g_1, \ldots)$ be the sequence selected by the cost-greedy algorithm. Fix some $K > 0$. Let $B = \sum_{i=1}^{K} c(g_i)$. Let F be (γ, δ)-approximately submodular as in Definition 4.2.1. For any sequence S and total cost C,*

$$F(\mathcal{G}_{\langle B \rangle}) > \left(1 - e^{-\gamma \frac{B}{C}}\right) \left(F(\mathcal{S}_{\langle C \rangle}) - \frac{\delta}{\gamma}\right).$$

Or more loosely:

$$F(\mathcal{G}_{\langle B \rangle}) > \left(1 - e^{-\gamma \frac{B}{C}}\right) F(\mathcal{S}_{\langle C \rangle}) - \delta \frac{B}{C}.$$

Proof. Define $\Delta_j = F(\mathcal{S}_{\langle C \rangle}) - F(\mathcal{G}_j) - \frac{\delta}{\gamma}$. By Lemma 4.2.3, $F(\mathcal{S}_{\langle C \rangle}) \leq F(\mathcal{G}_j) + \frac{C s_{j+1}}{\gamma} + \frac{\delta}{\gamma}$. By definition of s_{j+1}:

$$\Delta_j \leq \frac{C s_{j+1}}{\gamma} = \frac{C}{\gamma} \left(\frac{\Delta_j - \Delta_{j+1}}{c(g_j)}\right).$$

Rearranging we get $\Delta_{j+1} \leq \Delta_j (1 - \frac{c(g_{j+1})\gamma}{C})$. Unroll to get

$$\Delta_K \leq \Delta_0 \left(\prod_{j=1}^{K} 1 - \frac{c(g_j)\gamma}{C}\right).$$

Given that $B = \sum_{i=1}^{K} c(g_j)$, this is maximized at $c(g_j) = \frac{B}{K}$. Substituting in and using the fact that $\left(1 - \frac{z}{K}\right)^K < e^{-z}$:

$$\begin{aligned} \left(F(\mathcal{S}_{\langle C \rangle}) - \frac{\delta}{\gamma}\right) - F(\mathcal{G}_K) = \Delta_K &\leq \Delta_0 \left(1 - \gamma \frac{B}{C} \frac{1}{K}\right)^K \\ &< \left(F(\mathcal{S}_{\langle C \rangle}) - \frac{\delta}{\gamma}\right) e^{-\gamma \frac{B}{C}}, \end{aligned}$$

or $F(\mathcal{G}_{\langle B \rangle}) > \left(1 - e^{-\gamma \frac{B}{C}}\right) \left(F(\mathcal{S}_{\langle C \rangle}) - \frac{\delta}{\gamma}\right)$.

Since $\left(1 - e^{-\gamma\frac{B}{C}}\right)\frac{1}{\gamma} < \frac{B}{C}$, we can also write this as

$$F(\mathcal{G}_{\langle B\rangle}) > (1 - e^{-\gamma\frac{B}{C}})\left(F(\mathcal{S}_{\langle C\rangle})\right) - \delta\frac{B}{C}.$$

■

└

4.3 Approximate Greedy Maximization

In many cases, it is desirable to use an algorithm that does not actually implement the greedy strategy, but instead is *approximately greedy*. That is, the algorithm attempts to select an element that will significantly improve the value of the selected sequence, but does not always select the item that maximizes the greedy gain given in Equation (4.3).

For example, this type of algorithm is useful when searching over the entire set $\mathcal{X}$ for the maximum gain is prohibitively expensive, but a reasonably good element can be selected much more efficiently. Other examples include settings where the submodular function F is only able to be evaluated at training time, and a predictor is trained using contextual features to approximate F for use at test time [Streeter and Golovin, 2008, Ross et al., 2013]. A final example is the Orthogonal Matching Pursuit algorithm which we will examine in Chapter 5.

We now give a specific characterization of what it means to be approximately greedy so we can further analyze these algorithms.

Definition 4.3.1 (Approximately Greedy)**.** *Let $\mathcal{G}'_{j-1}$ be the set of elements selected by some algorithm $\mathcal{A}$ through $j-1$ iterations. Given a set function F, we say that $\mathcal{A}$ is approximately greedy if for all j there exists constants $\alpha_j \in [0,1]$ and $\beta_j \geq 0$ such that g'_j, the element selected by $\mathcal{A}$ at iteration j, satisfies:*

$$\frac{F(\mathcal{G}'_{j-1} \cup \{g'_j\}) - F(\mathcal{G}'_{j-1})}{c(g'_j)} \geq \max_{x\in\mathcal{X}} \frac{\alpha_j\left[F(\mathcal{G}'_{j-1} \cup \{x\}) - F(\mathcal{G}'_{j-1})\right] - \beta_j}{c(x)}.$$

For the special case when there exists $\alpha \leq \alpha_j$ and $\beta \geq \beta_j$ for all j for some psuedo-greedy algorithm $\mathcal{A}$, we say that $\mathcal{A}$ is (α,β)-approximately greedy.

Extending the previous approximation result for the greedy algorithm, we can get a similar bound for approximately greedy algorithms applied to approximately submodular function optimization. We first need to generalize Lemma 4.2.3 to also include the additive and multiplicative error introduced by the approximately greedy algorithm.

Lemma 4.3.2. *Let s'_j be the value*

$$\frac{F(\mathcal{G}'_{j-1} \cup \{g'_j\}) - F(\mathcal{G}'_{j-1})}{c(g'_j)},$$

for element g'_j selected by an approximately greedy algorithm according to Definition 4.3.1. Then for all sequences S and total costs C:

$$F(\mathcal{S}_{\langle C \rangle}) \leq F(\mathcal{G}'_{j-1}) + \frac{Cs'_j}{\alpha_j \gamma} + \frac{\beta_j}{\alpha_j \gamma} + \frac{\delta}{\gamma}.$$

Proof. By Lemma 4.2.2 we have:

$$\begin{aligned}
\alpha_j \frac{\gamma \left[F(\mathcal{G}'_{j-1} \cup \mathcal{S}_{\langle C \rangle}) - F(\mathcal{G}'_{j-1})\right] - \delta}{c(\mathcal{S}_{\langle C \rangle})} - \frac{\beta_j}{C} &\leq \alpha_j \max_{x \in \mathcal{S}_{\langle C \rangle}} \left[\frac{F(\mathcal{G}'_{j-1} \cup \{x\}) - F(\mathcal{G}'_{j-1})}{c(x)}\right] - \frac{\beta_j}{C} \\
&\leq \max_{x \in \mathcal{S}_{\langle C \rangle}} \frac{\alpha_j \left[F(\mathcal{G}'_{j-1} \cup \{x\}) - F(\mathcal{G}'_{j-1})\right] - \beta_j}{c(x)} \\
&\leq s'_j
\end{aligned}$$

By monotonicity we have $F(\mathcal{S}_{\langle C \rangle}) \leq F(\mathcal{G}_{j-1} \cup \mathcal{S}_{\langle C \rangle})$, and by definition $c(\mathcal{S}_{\langle C \rangle}) \leq C$ giving:

$$F(\mathcal{S}_{\langle C \rangle}) \leq F(\mathcal{G}'_{j-1}) + \frac{Cs'_j}{\alpha_j \gamma} + \frac{\beta_j}{\alpha_j \gamma} + \frac{\delta}{\gamma}.$$

■

Now, we can reproduce the same argument as used for analyzing the greedy algorithm, but with the previous lemma as a starting point.

Theorem 4.3.3. *Let $G' = (g'_1, \ldots)$ be the sequence selected by an (α, β)-approximately greedy algorithm as in Definition 4.3.1. Fix some $K > 0$. Let $B = \sum_{i=1}^{K} c(g'_i)$. Let F be (γ, δ)-approximately submodular as in Definition 4.2.1. For any sequence S and total cost C,*

$$F(\mathcal{G}_{\langle B \rangle}) > (1 - e^{-\alpha\gamma\frac{B}{C}}) \left(F(\mathcal{S}_{\langle C \rangle})\right) - \beta\frac{B}{C} - \alpha\delta\frac{B}{C}.$$

Proof. Define $\Delta_j = F(\mathcal{S}_{\langle C \rangle}) - F(\mathcal{G}_j)$. By Lemma 4.3.2, $F(\mathcal{S}_{\langle C \rangle}) \leq F(\mathcal{G}_j) + \frac{Cs_{j+1}}{\alpha_{j+1}\gamma} + \frac{\beta_{j+1}}{\alpha_{j+1}\gamma} + \frac{\delta}{\gamma}$.

By definition of s_{j+1}:

$$\Delta_j \leq \frac{Cs_{j+1}}{\alpha_{j+1}\gamma} + \frac{\beta_{j+1}}{\alpha_{j+1}\gamma} + \frac{\delta}{\gamma} = \frac{C}{\alpha_{j+1}\gamma}\left(\frac{\Delta_j - \Delta_{j+1}}{c(g_j)}\right) + \frac{\beta_{j+1}}{\alpha_{j+1}\gamma} + \frac{\delta}{\gamma}.$$

Rearranging we get $\Delta_{j+1} \leq \Delta_j(1 - \frac{c(g_{j+1})\alpha_{j+1}\gamma}{C}) + \frac{\beta_{j+1}c(g_{j+1})}{C} + \frac{\alpha_{j+1}\delta c(g_{j+1})}{C}$. Unroll to get

$$\Delta_K \leq \Delta_0 \left(\prod_{j=1}^{K} 1 - \frac{c(g_j)\alpha_j\gamma}{C}\right) + \sum_{j=1}^{K}\left(\prod_{i=j+1}^{K} 1 - \frac{c(g_i)\alpha_i\gamma}{C}\right)\left(\frac{\beta_j c(g_j)}{C} + \frac{\alpha_j\delta c(g_j)}{C}\right).$$

Using $1 - \frac{c(g_j)\alpha_j\gamma}{C} < 1$,

$$\Delta_K \leq \left(\Delta_0 + \sum_{j=1}^{K}\frac{\beta_j c(g_j)}{C} + \sum_{j=1}^{K}\frac{\alpha_j\delta c(g_j)}{C}\right)\left(\prod_{j=1}^{K} 1 - \frac{c(g_j)\alpha_j\gamma}{C}\right).$$

Let $\alpha < \alpha_j$. Given that $B = \sum_{i=1}^{K} c(g_j)$, this is maximized at $c(g_j) = \frac{B}{K}$. Substituting in and using the fact that $\left(1 - \frac{z}{K}\right)^K < e^{-z}$:

$$\begin{aligned} F(\mathcal{S}_{\langle C\rangle}) - F(\mathcal{G}_K) = \Delta_K &\leq \left(\Delta_0 + \sum_{j=1}^{K}\beta_j\frac{c(g_j)}{C} + \alpha\delta\frac{B}{C}\right)\left(1 - \alpha\gamma\frac{B}{C}\frac{1}{K}\right)^K \\ &< \left(F(\mathcal{S}_{\langle C\rangle}) + \sum_{j=1}^{K}\beta_j\frac{c(g_j)}{C} + \alpha\delta\frac{B}{C}\right)e^{-\alpha\gamma\frac{B}{C}}, \end{aligned}$$

or $F(\mathcal{G}_{\langle B\rangle}) > \left(1 - e^{-\alpha\gamma\frac{B}{C}}\right)\left(F(\mathcal{S}_{\langle C\rangle}) - \sum_{j=1}^{K}\beta_j\frac{c(g_j)}{C} - \alpha\delta\frac{B}{C}\right)$.

Since $\left(1 - e^{-\alpha\gamma\frac{B}{C}}\right) < 1$, we can also write this as

$$F(\mathcal{G}_{\langle B\rangle}) > (1 - e^{-\alpha\gamma\frac{B}{C}})\left(F(\mathcal{S}_{\langle C\rangle})\right) - \sum_{j=1}^{K}\beta_j\frac{c(g_j)}{C} - \alpha\delta\frac{B}{C}.$$

Now, if $\beta > \beta_j$ for all j,

$$F(\mathcal{G}_{\langle B\rangle}) > (1 - e^{-\alpha\gamma\frac{B}{C}})\left(F(\mathcal{S}_{\langle C\rangle})\right) - \beta\frac{B}{C} - \alpha\delta\frac{B}{C}.$$

■

As expected from the greedy algorithm analysis, the multiplicative and additive terms (α, β) get incorporated in the same manner as the respective (γ, δ) terms from the approximate submodularity

bound.

This bound incorporates a number of features of previous results as well. For example, the corresponding bound from the work of Das and Kempe [2011] for the Orthogonal Matching Pursuit algorithm in the unit-cost case has the same $\alpha\gamma$ term.

Other work on no-regret learning of approximators for the submodular function F [Streeter and Golovin, 2008, Ross et al., 2013] incorporates the additive term

$$\sum_{j=1}^{J} \beta_j \frac{c(g_j)}{C}$$

where β_j is the error made by the no-regret learner at each iteration. This same term is seen in the previous proof, and is only simplified using $\beta < \beta_j$. The same analysis above could be used to extend similar no-regret analyses to the approximately submodular setting, in the same manner as this previous work.

4.4 Bi-criteria Approximation Bounds for Arbitrary Budgets

In the previous sections, we derived bounds that hold only for the budgets B which correspond to the budgets at which the algorithm adds a new element to the sequence. In many settings, this guarantee can be poor for the list selected by the greedy algorithm in practice. For example, if the algorithm selects a single, high-cost, high-value item at the first iteration, then the smallest budget the guarantee holds for is the cost of that item.

As discussed in Section 4.1, there are many approaches that can obtain guarantees for any arbitrary budget B, but unfortunately these algorithms do not generate a single common sequence for all budgets. Because we ultimately want anytime or anybudget behavior, we would like similar guarantees for a budget agnostic algorithm. Unfortunately, as we will show shortly, a guarantee that has the same form as previous ones is not possible for arbitrary submodular functions. Instead, in this section we will develop an algorithm and corresponding bound that gives a *bi-criteria approximation* in both value and budget. Such a bound will have the form

$$F(\mathcal{G}_{\langle B \rangle}) > (1 - c_1) F(\mathcal{S}_{\langle \frac{B}{c_2} \rangle}) - \dots . \tag{4.4}$$

Here we have the standard $(1 - c_1)$-approximation in value when compared to any arbitrary sequence S, but we also have a c_2-approximation in budget, that is, in order to be competitive with a given sequence we need to incur c_2 additional cost.

We will now show the inherent difficulty in obtaining good performance from a budget agnostic algorithm which generates a single sequence, and demonstrate the necessity of the bi-criteria approximation given above. Consider the following budgeted maximization problem:

$$\begin{aligned} \mathcal{X} = \{1, 2, \dots\}, \qquad & c(x) = x \\ F(\mathcal{S}) = \sum_{x \in \mathcal{S}} & e^x. \end{aligned} \tag{4.5}$$

We can use this problem to illustrate the inherent difficulty in generating single sequences that are competitive at arbitrary budgets B. This problem is in fact a modular optimization, and furthermore, has a very simple optimal solution of always selecting the single largest element that fits within a given budget. As the next result shows, however, even achieving a bound derived for submodular functions is difficult unless the cost approximation is fairly loose.

Theorem 4.4.1. *Let $\mathcal{A}$ be any algorithm for selecting sequences $A = (a_1, \ldots)$. The best bi-criteria approximation $\mathcal{A}$ can satisfy must be at least a 4-approximation in cost for the sequence described in Equation (4.5). That is, there does not exist a $C < 4$ such that, for all $B > c_{min}$ and all sequences S,*

$$F(\mathcal{A}_{\langle B \rangle}) > \left(1 - \frac{1}{e}\right) F(\mathcal{S}_{\langle \frac{B}{C} \rangle})$$

Proof. First, by construction of the problem, it is clear that the optimal set for a given (integral) budget B' is to select the largest element $x < B'$ for a value of $e^{B'}$. Furthermore, because

$$\frac{\sum_{x=1}^{B'-1} e^x}{e^{B'}} \leq \frac{1}{e-1},$$

the only way to be a $(1 - \frac{1}{e})$-approximation in value for all sets $\mathcal{S}_{\langle B' \rangle}$ is to have selected an element $x \geq B'$.

Consider the sequence A at some element j. Let the cost $c(\mathcal{A}_j) = b$. Let the largest element in $\mathcal{A}_j$ be some function of b, $f(b)$ with value $e^{f(b)}$.

Now consider the next element a_{j+1}. To maintain the property that A is a C-approximation in cost, $c(\mathcal{A}_{j+1})$ is at most $Cf(b)$, which implies that $c(a_{j+1}) \leq Cf(b) - b$. In order for the sequence to continue extending itself to arbitrary large budgets, the ratio between the cost of the sequence and the largest element in the sequence must be increasing, giving

$$\frac{b}{f(b)} \leq \frac{Cf(b)}{Cf(b) - b}.$$

Rearranging and using the fact that all terms are positive gives

$$b^2 - Cbf(b) + Cf(b)^2 \geq 0.$$

The above inqeuality only holds when $\sqrt{C} \geq 2$, or $C \geq 4$, proving the theorem. ∎

As an aside, the argument in the proof above also shows that the optimal single sequence A, *i.e.*the sequence for which the argument above is tightest, will output elements a_j with cumulative cost $c(\mathcal{A}_j) = b$ such that $c(a_j) = f(b) = \frac{b}{\sqrt{C}}$. In the case of the tightest achievable bound when $C = 4$, this corresponds to selecting an element at every iteration that roughly doubles the current

cost of the list. We will examine an algorithm (Algorithm 4.2) which demonstrates exactly this doubling behavior.

Now that we have an upper bound on the best cost-approximation any single sequence algorithm can obtain, we now present an algorithm which does satisfy the bi-criteria approximation in Equation (4.4), with a cost approximation factor of 6. [1]

Algorithm 4.2 presents a doubling strategy for selecting a sequence of elements by effectively doubling the space of elements that can be added at each iteration. At the first iteration, the algorithm selects elements less than some minimum cost $c_{\min}$. For every following iteration, the algorithm selects from all elements with cost less than the total cost of the items selected so far, at most doubling the total cost of the sequence.

Algorithm 4.2 Doubling Algorithm

Given: objective function F, elements $\mathcal{X}$, minimum cost $c_{\min}$.
Let $G_1 = \underset{x \in \mathcal{X},\ c(x) \leq c_{\min}}{\arg\max} \frac{F(\{x\})}{c(x)}$.
Let $\mathcal{G}_1 = \{g_1\}$.
for $j = 2, \ldots$ **do**
  Let $g_j = \underset{x \in \mathcal{X} \setminus \mathcal{G}_{j-1},\ c(x) \leq c(\mathcal{G}_{j-1})}{\arg\max} \frac{F(\mathcal{G}_{j-1} \cup \{x\}) - F(\mathcal{G}_{j-1})}{c(x)}$.
  Let $\mathcal{G}_j = \mathcal{G}_{j-1} \cup \{g_j\}$.
end for

This algorithm allows for a bi-criteria approximation for arbitrary approximately submodular maximization problems, as long as the doubling algorithm doesn't get stuck at any iteration. The following definition just outlines the conditions that allow the doubling algorithm to suceed, namely that the algorithm can always continue to select new elements at every iteration.

Definition 4.4.2. *Let $G = (g_1, \ldots)$ be the sequence selected by the doubling algorithm. The set $\mathcal{X}$ and function F are doubling capable if, at every iteration j, the set*

$$\{x \mid x \in \mathcal{X} \setminus \mathcal{G}_{j-1},\ c(x) \leq c(\mathcal{G}_{j-1})\}$$

is non-empty.

We will assume that this definition holds for the rest of the analysis. In order to prove the bi-criteria approximation, we first need the following lemma, describing the behavior of the total cost of the subsets selected by the doubling algorithm.

[1] We conjecture that the cost approximation factor for Algorithm 4.2 is actually 4, but are not able to prove it directly using the analysis here.

Lemma 4.4.3. *Let $G = (g_1, \ldots)$ be the sequence selected by the doubling algorithm. Fix some B such that $c_{min} < B < c(\mathcal{X})$. There exists some K such that $\frac{B}{2} \leq \sum_{i=1}^{K} c(g_i) \leq B$.*

Proof. Consider the largest K such that $\sum_{i=1}^{K} c(g_i) \leq B$. Examine the just element, g_{K+1}. Both g_K and g_{K+1} must exist because $\mathcal{X}$ is doubling capable according to Definition 4.4.2. By construction,

$$\sum_{i=1}^{K+1} c(g_i) \geq B.$$

Because $c(g_{K+1}) \leq c(\mathcal{G}_K)$ we know that

$$2\sum_{i=1}^{K} c(g_i) \geq \sum_{i=1}^{K+1} c(g_i) \geq B,$$

completing the proof. ■

Using the above lemma, we can now give the bi-criteria approximation bound for Algorithm 4.2 for any given budget. The bound gives a 6 approximation in cost and the same approximation in value as the previous greedy result in Theorem 4.2.4.

Theorem 4.4.4. *Let $G = (g_1, \ldots)$ be the sequence selected by the doubling algorithm (Algorithm 4.2). Fix some $B > c_{min}$. Let F be (γ, δ)-approximately submodular as in Definition 4.2.1. For any sequence S,*

$$F(\mathcal{G}_{\langle B \rangle}) > \left(1 - e^{-\gamma}\right) F(\mathcal{S}_{\langle \frac{B}{6} \rangle}) - \delta.$$

Proof. Clearly, if $B \geq c(\mathcal{X})$, the theorem trivially holds for $\mathcal{G}_{\langle B \rangle} = \mathcal{X}$.

If $B \leq c(\mathcal{X})$, using Lemma 4.4.3, we know that there must be some K such that $\frac{B}{2} \leq \sum_{i=1}^{K} c(g_i) \leq B$. Similiarly there must exist some k such that $\frac{B}{6} \leq \sum_{i=1}^{k} c(g_i) \leq \frac{B}{3}$.

Consider the sequence $G' = (g_{k+1}, \ldots, g_K)$. Let $\mathcal{G}'_j = \mathcal{G}_k \cup \{g_{k+1}, \ldots, g_j\}$. We can derive a modified version of the bound in Lemma 4.2.3 that gives:

$$F(\mathcal{S}_{\langle \frac{B}{6} \rangle}) \leq F(\mathcal{G}'_{j-1}) + \frac{Cs'_j + \delta}{\gamma},$$

where s'_j is the maximum gain at iteration $j+k$ of Algorithm 4.2. This holds because $c(\mathcal{G}_k) \geq \frac{B}{6}$, implying that the maximum at iteration $k + j$ used to calculate s'_j is over a superset of the elements in $\mathcal{S}_{\langle \frac{B}{6} \rangle}$.

Now, using that modified version of Lemma 4.2.3, we can re-apply the same argument from

Theorem 4.2.4 and show that

$$\left(F(\mathcal{S}_{\langle \frac{B}{6} \rangle}) - \frac{\delta}{\gamma}\right) - F(\mathcal{G}_K) \leq \left(F(\mathcal{S}_{\langle \frac{B}{6} \rangle}) - \frac{\delta}{\gamma}\right) \left(\prod_{j=k}^{K} 1 - \frac{6c(g_j)\gamma}{B}\right)$$

By construction of k and K, $\sum_{j=k}^{K} c(g_j) \geq \frac{B}{6}$, so we can simplify this to

$$\left(F(\mathcal{S}_{\langle \frac{B}{6} \rangle}) - \frac{\delta}{\gamma}\right) - F(\mathcal{G}_K) \leq \left(F(\mathcal{S}_{\langle \frac{B}{6} \rangle}) - \frac{\delta}{\gamma}\right) \left(1 - \gamma \frac{1}{K}\right)^K,$$

after which bounding the $(1 - \frac{\gamma}{K})^K$ term and rearranging gives the final bound:

$$F(\mathcal{G}_{\langle B \rangle}) > (1 - e^{-\gamma}) \left(F(\mathcal{S}_{\langle \frac{B}{6} \rangle})\right) - \delta.$$

■

The same basic arguments as above can be also be used to show that an approximately greedy algorithm, when modified with the selection strategy of the doubling algorithm, will also give a bi-criteria approximation for any budget. The cost approximation will still be a factor of 6, and the value approximation will be the same as in Theorem 4.3.3.

Chapter 5

Sparse Approximation

In this chapter we will examine the sparse approximation or subset selection problem. We will show that the greedy and approximately greedy algorithm analysis of the previous chapter can be applied here, and will derive corresponding approximation results for this setting. In constrast to previous work on similar analyses, we derive bounds that depend primarily on factors which place small weights on the optimal subset of features, as opposed to previous work, where the approximation bounds depend primarily on factors related to the geometry, or orhogonality, of the features. We also present novel, time-based versions of classic feature or subset selection algorithms, and show that for the budgeted feature selection problem, these approaches significantly outperform approaches which do not consider feature cost.

5.1 Background

Given a set of variables or features $X_i \in \mathcal{X}$ and a target variable Y, the sparse approximation problem is to select a subset of the variables $\mathcal{D} \subset \mathcal{X}$ that minimizes the reconstruction error

$$\min_w \mathbb{E}[\frac{1}{2}(Y - w^T X_{\mathcal{D}})^2],$$

where $X_{\mathcal{S}} = [X_i | X_i \in \mathcal{S}]$. Typically the selection is done with respect to some constraint on the selected $\mathcal{D}$, such as a cardinality constraint $|\mathcal{D}| \leq B$.

This problem is commonly framed in the literature as a constrained loss minimization problem of the loss function

$$f(w) = \mathbb{E}[(Y - w^T X)^2], \tag{5.1}$$

where the constaint is designed to induce sparsity on the weight vector w.

The sparse approximation problem can then be written as a loss minimization with respect to a constraint on the number of non-zero entries in w:

$$\begin{matrix} \min_w f(w) \\ \|w\|_0 \leq B \end{matrix}, \tag{5.2}$$

Algorithm 5.1 Forward Regression

Given: elements $\mathcal{X}$, target $\mathcal{Y}$
Define F as in Equation (5.3)
Let $\mathcal{D}_0 = \emptyset$.
for $j = 1, \ldots$ **do**
  Let $x^* = \arg\max_{x \in \mathcal{X}} \frac{F(\mathcal{D}_{j-1} \cup \{x\}) - F(\mathcal{D}_{j-1})}{c(x)}$.
  Let $\mathcal{D}_j = \mathcal{D}_{j-1} \cup \{x^*\}$.
end for

where $\|\|_0$ is the "0-norm", or number of non-empty elements in w. The selected elements in $\mathcal{D}$ then simply correspond to the non-zero indices selected in the optimal solution to the above constrained problem.

Another way to re-write the sparse approximation objective is as a set function $F(\mathcal{S})$:

$$F(\mathcal{S}) = \frac{1}{2}\mathbb{E}[Y^2] - \min_{w \in \mathbb{R}^{|\mathcal{S}|}} \frac{1}{2}\mathbb{E}[(Y - w^T X_{\mathcal{S}})^2] \tag{5.3}$$

$$= \max_{w \in \mathbb{R}^{|\mathcal{S}|}} b_{\mathcal{S}}^T w - \frac{1}{2} w^T C_{\mathcal{S}} w, \tag{5.4}$$

where b is a vector of covariances such that $b_i = \text{Cov}(X_i, Y)$ and C is the covariance matrix of the variables X_i, with $C_{ij} = \text{Cov}(X_i, X_j)$. Furthermore, $C_{\mathcal{S}}$ is the subset of rows and columns of C corresponding to the variables selected in $\mathcal{S}$ and $b_{\mathcal{S}}$ is the equivalent over vector indices of the vector b.

The sparse approximation problem is then the same as the monotone maximization problem over the set function F given in Equation (4.2), as studied in the previous chapter. Extending this problem to the budgeted setting from the previous chapter as well, each feature also has an associated cost $c(X_i)$ and the goal is to select a subset $\mathcal{D}$ such that $c(\mathcal{D}) = \sum_{x \in \mathcal{D}} c(x) < B$. For the coordinate-based version of the problem in Equation (5.2), this can be replaced with a weighted equivalent of the zero-norm.

For our analysis, we will stick to the set function maximization setting analyzed in Chapter 4, but these representations are all equivalent.

5.1.1 Algorithms

Two approaches to solving this problem which we will analyze here are two cost-greedy versions of the existing the Forward Regression [Miller, 2002] and Orthogonal Matching Pursuit Pati et al. [1993] algorithms. The cost-aware Forward Regression algorithm, given in Algorithm 5.1 simply selects the next variable x which maximizes the gain in objective F, divided by the cost $c(x)$. This is equivalent to the standard cost-greedy algorithm (Algorithm 4.1) for maximizing the set function F given in Equation (5.3).

Algorithm 5.2 Orthogonal Matching Pursuit

Given: elements $\mathcal{X}$, target $\mathcal{Y}$
Define F as in Equation (5.3)
Let $\mathcal{D}_0 = \emptyset$.
for $j = 1, \ldots$ **do**
  Let $w^* = \arg\min_w \mathbb{E}[(Y - w^T X_{\mathcal{D}_{j-1}})^2]$
  Let $x^* = \arg\max_{x \in \mathcal{X}} \frac{\mathbb{E}[(Y - w^{*T} X_{\mathcal{D}_{j-1}})^T x]^2}{c(x)}$.
  Let $\mathcal{D}_j = \mathcal{D}_{j-1} \cup \{x^*\}$.
end for

The Orthogonal Matching Pursuit (OMP) algorithm [Pati et al., 1993], modified to handle feature costs, given in Algorithm 5.2 is a more specialized algorithm for optimizing set functions F that correspond to an underlying loss optimization. The classic OMP algorithm for optimizing a loss function first computes a gradient of the loss function given the currently selected variables, then selects the next variable which maximizes the inner product with the computed gradient. In the sparse approximation setting, this corresponds to computing the current residual $Z = Y - w^{*T} X_{\mathcal{D}}$ given the currently selected elements $\mathcal{D}$, and then selecting X_i which maximizes $(\text{Cov}(X_i, Z))^2$, or $\mathbb{E}[(Y - w^{*T} X_{\mathcal{D}_{j-1}})^T X_i]^2$. To make this algorithm cost-aware, we simply augment the greedy maximization of the gradient term to be discounted by the feature cost.

For the coordinate-based version of the sparse approximation problem (Equation (5.2)), this can also be viewed as performing a coordinate descent over the weight vector w, as the maximization

$$x^* = \arg\max_{x \in \mathcal{X}} \frac{|\mathbb{E}[(Y - w^{*T} X_{\mathcal{D}_{j-1}})x]|}{c(x)}$$

is equivalent to selecting the dimension of w with the corresponding steepest gradient.

In previous work [Das and Kempe, 2011], these algorithms, applied to the sparse approximation problem, have been analyzed in the context of the submodular optimization setting. This previous work has shown that the sparse approximation problem is in fact approximately submodular (Definition 4.2.1), and that the submodular optimization analysis shown in the last chapter can be directly applied to these algorithms and the sparse approximation problem.

Specifically, they show that the sparse approximation problem is $(\lambda_{\min}(C), 0)$-approximately submodular, where $\lambda_{\min}$ is the minimum eigenvalue of the covariance matrix C, which captures the degree to which the variables X_i are non-orthogonal. Additionally, they show that the OMP algorithm is also $(\lambda_{\min}(C), 0)$-approximately greedy in this setting.

Let $\mathcal{D}_{\langle B \rangle}$ be the set selected by forward regression for some budget B, and $\mathcal{S}_{\langle C \rangle}$ be the optimal set of features for some other budget C. Using the results in Theorem 4.2.4 this previous work gives a bound of

$$F(\mathcal{D}_{\langle B \rangle}) > \left(1 - e^{-\lambda_{\min}(C)\frac{B}{C}}\right) F(\mathcal{S}_{\langle C \rangle})$$

on the performance of the forward algorithm compared to optimal performance. Similarly for the OMP algorithm we find an approximation factor of

$$\left(1 - e^{-\lambda_{\min}(C)^2 \frac{B}{C}}\right).$$

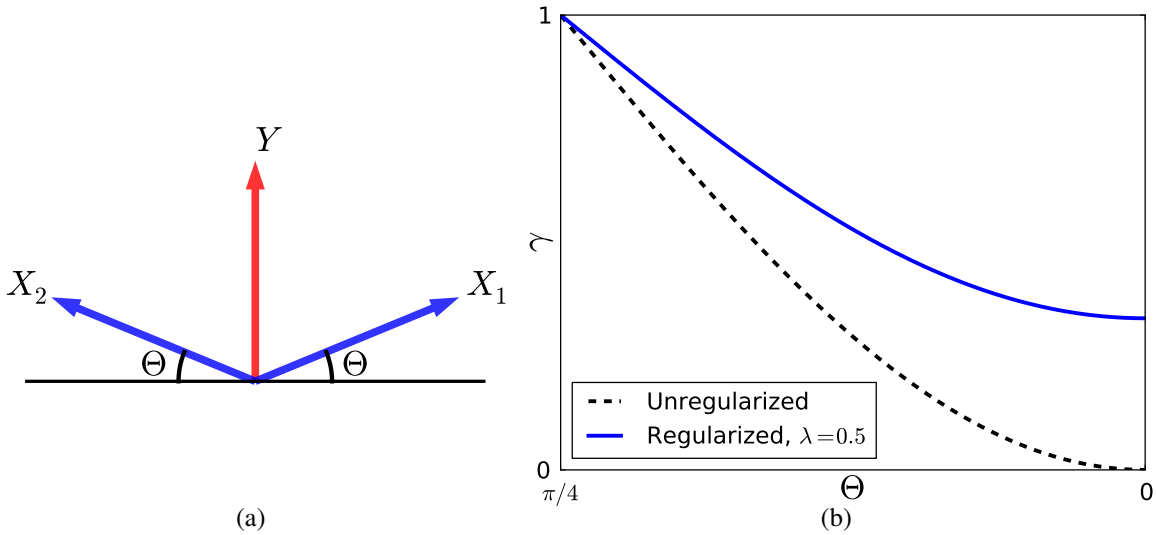


Figure 5.1: (a) Non-submodularity in features– that is when the sum is better then the parts– occurs when two correlated features can be combined to reach a target that each poorly represents on their own. This occurs often in practice; however, it takes very large weights for the combination of features to be better then the features taken alone, which is disallowed by regularization. (b) Illustration of the approximation guarantee for a simple problem with two highly correlated features, as a function of the correlation, or angle, between the two features. We illustrate the bound for the completely spectral case [Das and Kempe, 2011], and for the same problem with regularization of $\lambda = 0.5$ using the bound presented.

In many settings, however, these geometric factors can approach their worst case bounds. Just two highly correlated features can cause the minimum eigenvalue of C to be extremely small. For example, in future chapters, we will be applying this same analysis to sets of "features" X_i that are the outputs of a set of weak predictors, for example the outputs of all decision trees defined over a set of training points. In this setting the bounds are extremely weak, as minor changes to a given weak predictor will produce another highly correlated "feature" in the set $\mathcal{X}$.

Intuitively, the geometric factors that previous bounds have relied on are needed for analysis because as the variables involved diverge from orthogonality and become more dependent, the performance of multiple vectors combined together can vastly outperform the performance of each vector in isolation. This gap between the combined gain and individual gain for a set of elements makes greedy selection perform arbitrarily poorly compared to combinatorial enumeration, as the bounds in Chapter 4 and previous results [Krause and Cehver, 2010, Das and Kempe, 2011] show.

The inherent difficulty in the subset selection problem when the variables are highly dependent is due to a simple fact that can be illustrated geometrically: when two vectors are nearly parallel, combining them together with large weights can produce new vectors that are nearly orthogonal to either of the two original vectors. This in turn can cause two variables to have small individual gains, while still having large combined gains and in turn weakening the approximate submodularity guarantees of the problem.

These problems with non-orthogonality only arise in the presence of large weighted combinations of the underlying variables. To analyze the impact that the magnitude of the weights has on the resulting approximation bounds, we will now analyze two regularized variants of the sparse approximation problem. Using regularization we can reduce the impact that large weight vectors can have on the gain of any given subset, thereby improving the approximate submodularity-based bounds.

Furthermore, it is typically beneficial in practice to use some amount of regularization, to avoid overfitting and increase the robustness of a selected set. In the abscence of exponentially large amounts of training data, a small amount of regularization would be warranted anyway, so any improvement in the theoretical guarantees of the algorithm is just another added benefit.

5.2 Regularized Sparse Approximation

The first approach to regularization we will analyze is a Tikhonov regularized version of the problem which will directly penalize the gain of large weight vectors. This regularized version of the sparse approximation problem is given as

$$F(\mathcal{S}) = \frac{1}{2}\mathbb{E}[Y^2] - \min_{w \in \mathbb{R}^{|\mathcal{S}|}} \frac{1}{2}\mathbb{E}[(Y - w^T X_{\mathcal{S}})^2 + \lambda w^T w] \tag{5.5}$$

$$= \max_{w \in \mathbb{R}^{|\mathcal{S}|}} b_{\mathcal{S}}^T w - \frac{1}{2} w^T (C_{\mathcal{S}} + \lambda I) w, \tag{5.6}$$

where b and C are the covariance vector and matrix as defined previously.

Just as in previous work [Das and Kempe, 2011] for the unregularized case, we can show that this regularized version of the sparse approximation problem is approximately submodular as given in Definition 4.2.1.

To do this, we will first need to consider a few lemmas which allow us to relate our result to the spectral properties used in previous work [Das and Kempe, 2011].

Let $C_{\mathcal{S}}^{\mathcal{A}}$ be the covariance matrix of the residual components of the set $\mathcal{S}$ with respect to the set $\mathcal{A}$. Specifically, if we define $\text{Res}(X_i, \mathcal{A})$ to be the portion of X_i orthogonal to the variables selected in $\mathcal{A}$, then $C_{\mathcal{S}}^{\mathcal{A}}$ is the covariance matrix of the variables $X_i' = \text{Res}(X_i, \mathcal{A})$ for $i \in \mathcal{S}$. The first lemma relates the eigenvalues of the covariance matrix of residuals, $C_{\mathcal{S}}^{\mathcal{A}}$ to the equivalent matrix that will appear in the analysis of the regularized problem.

Lemma 5.2.1. *Given sets of variables $\mathcal{S}$ and $\mathcal{A}$, let $C_{\mathcal{S}}^{\mathcal{A}}$ be the covariance matrix for the residual components of $\mathcal{S}$ with respect to $\mathcal{A}$, and $C_{\mathcal{S}}^{\mathcal{A}'}$ such that*

$$C_{\mathcal{S}}^{\mathcal{A}} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} C_{\mathcal{A}}^{-1} C_{\mathcal{A}\mathcal{S}}$$
$$C_{\mathcal{S}}^{\mathcal{A}'} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \lambda I\right)^{-1} C_{\mathcal{A}\mathcal{S}},$$

for some λ. Then

$$\lambda_{min}(C_{\mathcal{S}}^{\mathcal{A}'}) \geq \lambda_{min}(C_{\mathcal{S}}^{\mathcal{A}})$$

where $\lambda_{min}(C)$ is the minimum eigenvalue of C.

Proof. For all vectors x, we have that

$$x^T C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \lambda I\right)^{-1} C_{\mathcal{A}\mathcal{S}} x \leq x^T C_{\mathcal{S}\mathcal{A}} C_{\mathcal{A}}^{-1} C_{\mathcal{A}\mathcal{S}} x,$$

which implies that

$$x^T C_{\mathcal{S}}^{\mathcal{A}'} x \geq x^T C_{\mathcal{S}}^{\mathcal{A}} x$$

for all x.

Given that $\lambda_{\min}(C) = \min_{x^T x = 1} x^T C x$, we have that

$$\lambda_{\min}(C_{\mathcal{S}}^{\mathcal{A}'}) = \min_{x^T x = 1} x^T C_{\mathcal{S}}^{\mathcal{A}'} x \geq \min_{x^T x = 1} x^T C_{\mathcal{S}}^{\mathcal{A}} x = \lambda_{\min}(C_{\mathcal{S}}^{\mathcal{A}}),$$

completing the proof. ■

The next lemma is taken directly from previous work, and bounds the smallest eigenvalue of $C_{\mathcal{S}}^{\mathcal{A}}$ in terms of that of the whole covariance matrix C. For more details on the proof of this lemma, we refer the reader to the previous work.

Lemma 5.2.2 (Lemmas 2.5 and 2.6 from [Das and Kempe, 2011])**.** *Given sets of variables $\mathcal{S}$ and $\mathcal{A}$, let $C_{\mathcal{S}}^{\mathcal{A}}$ be the covariance matrix for the residual components of $\mathcal{S}$ with respect to $\mathcal{A}$, i.e.*

$$C_{\mathcal{S}}^{\mathcal{A}} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} C_{\mathcal{A}}^{-1} C_{\mathcal{A}\mathcal{S}}.$$

Then

$$\lambda_{min}(C_{\mathcal{S}}^{\mathcal{A}}) \geq \lambda_{min}(C)$$

where $\lambda_{min}(C)$ is the minimum eigenvalue of C. ■

Finally, the last spectral lemma we need is simply a bound on the relationship between the quadratic form $b^T Q^{-1} b$ and the norm $b^T b$.

Lemma 5.2.3. *Let b be an arbitrary vector and Q a positive definite matrix. Then*

$$b^T Q^{-1} b \leq \frac{b^T b}{\lambda_{min}(Q)}.$$

Proof. Adapting the argument from Das and Kempe [2011], we have:

$$\frac{b^T Q^{-1} b}{b^T b} \leq \max_v \frac{v^T Q^{-1} v}{v^T v} = \lambda_{\max}(Q) = \frac{1}{\lambda_{\min}(Q)}.$$

■

Now, to analyze the actual regularized sparse approximation problem, we can first derive an expression for the gain $F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})$, to be used in proving the later theorems.

Lemma 5.2.4. *For some X_i and Y, Let F be as given in Equation (5.5) with regularization parameter λ. Then*

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) = \frac{1}{2} b_{\mathcal{S}}^{\mathcal{A}'^T} (C_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'},$$

where

$$b_{\mathcal{S}}^{\mathcal{A}'} = b_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \lambda I\right)^{-1} b_{\mathcal{A}}$$
$$C_{\mathcal{S}}^{\mathcal{A}'} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \lambda I\right)^{-1} C_{\mathcal{A}\mathcal{S}}$$

Proof. Starting with the definition of F from Equation (5.5), we have

$$\max_w \left[b_{\mathcal{A}\cup\mathcal{S}}^T w - \frac{1}{2} w^T \left(C_{\mathcal{A}\cup\mathcal{S}} + \lambda I\right) w \right] - \max_v \left[b_{\mathcal{A}}^T v - \frac{1}{2} v^T \left(C_{\mathcal{A}} + \lambda I\right) v \right].$$

If we break the matrix $C_{\mathcal{A}\cup\mathcal{S}}$ up in to blocks:

$$C_{\mathcal{A}\cup\mathcal{S}} = \begin{bmatrix} C_{\mathcal{A}} & C_{\mathcal{A}\mathcal{S}} \\ C_{\mathcal{S}\mathcal{A}} & C_{\mathcal{S}} \end{bmatrix},$$

and similarly break up $b_{\mathcal{A}\cup\mathcal{S}}$, we can rewrite as:

$$
\begin{aligned}
&F(\mathcal{A}\cup\mathcal{S}) - F(\mathcal{A}) \\
&= \max_{w_\mathcal{S}, w_\mathcal{A}} \left[b_\mathcal{S}^T w_\mathcal{S} + 2 b_\mathcal{A} w_\mathcal{A} - \frac{1}{2} w_\mathcal{S}^T \left(C_\mathcal{S} + \lambda I\right) w_\mathcal{S} - w_\mathcal{A}^T C_{\mathcal{A}\mathcal{S}} w_\mathcal{S} - \frac{1}{2} w_\mathcal{A}^T \left(C_\mathcal{A} + \lambda I\right) w_\mathcal{A} \right] \\
&\quad - \max_v \left[b_\mathcal{A}^T v - \frac{1}{2} v^T \left(C_\mathcal{A} + \lambda I\right) v \right] \\
&= \max_{w_\mathcal{S}} \left[b_\mathcal{S}^T w_\mathcal{S} - \frac{1}{2} w_\mathcal{S}^T \left(C_\mathcal{S} + \lambda I\right) w_\mathcal{S} + \max_{w_\mathcal{A}} \left[\left(b_\mathcal{A} - C_{\mathcal{A}\mathcal{S}} w_\mathcal{S}\right)^T w_\mathcal{A} - \frac{1}{2} w_\mathcal{A}^T \left(C_\mathcal{A} + \lambda I\right) w_\mathcal{A} \right] \right. \\
&\quad \left. - \max_v \left[b_\mathcal{A}^T v - \frac{1}{2} v^T \left(C_\mathcal{A} + \lambda I\right) v \right] \right].
\end{aligned}
$$

Solving for the optimizations over $\mathcal{A}$ directly gives

$$
\begin{aligned}
&= \max_{w_\mathcal{S}} \left[b_\mathcal{S}^T w_\mathcal{S} - \frac{1}{2} w_\mathcal{S}^T \left(C_\mathcal{S} + \lambda I\right) w_\mathcal{S} + \frac{1}{2} \left(b_\mathcal{A} - C_{\mathcal{A}\mathcal{S}} w_\mathcal{S}\right)^T \left(C_\mathcal{A} + \lambda I\right)^{-1} \left(b_\mathcal{A} - C_{\mathcal{A}\mathcal{S}} w_\mathcal{S}\right) \right. \\
&\quad \left. - \frac{1}{2} b_\mathcal{A}^T \left(C_\mathcal{A} + \lambda I\right)^{-1} b_\mathcal{A} \right] \\
&= \max_{w_\mathcal{S}} \left[b_\mathcal{S}^T w_\mathcal{S} - \frac{1}{2} w_\mathcal{S}^T \left(C_\mathcal{S} + \lambda I\right) w_\mathcal{S} + \frac{1}{2} w_\mathcal{S}^T C_{\mathcal{S}\mathcal{A}} \left(C_\mathcal{A} + \lambda I\right)^{-1} C_{\mathcal{A}\mathcal{S}} w_\mathcal{S} \right. \\
&\quad - \left(C_{\mathcal{S}\mathcal{A}} \left(C_\mathcal{A} + \lambda I\right)^{-1} b_\mathcal{A}\right)^T w_\mathcal{S} \\
&\quad \left. + \frac{1}{2} b_\mathcal{A}^T \left(C_\mathcal{A} + \lambda I\right)^{-1} b_\mathcal{A} - \frac{1}{2} b_\mathcal{A}^T \left(C_\mathcal{A} + \lambda I\right)^{-1} b_\mathcal{A} \right] \\
&= \max_{w_\mathcal{S}} \left[b_\mathcal{S}^{\mathcal{A}'^T} w_\mathcal{S} - \frac{1}{2} w_\mathcal{S}^T \left(C_\mathcal{S}^{\mathcal{A}'} + \lambda I\right) w_\mathcal{S} \right],
\end{aligned}
$$

where

$$
\begin{aligned}
b_\mathcal{S}^{\mathcal{A}'} &= b_\mathcal{S} - C_{\mathcal{S}\mathcal{A}} \left(C_\mathcal{A} + \lambda I\right)^{-1} b_\mathcal{A} \\
C_\mathcal{S}^{\mathcal{A}'} &= C_\mathcal{S} - C_{\mathcal{S}\mathcal{A}} \left(C_\mathcal{A} + \lambda I\right)^{-1} C_{\mathcal{A}\mathcal{S}}.
\end{aligned}
$$

Solving this completes the proof:

$$
\max_{w_\mathcal{S}} \left[b_\mathcal{S}^{\mathcal{A}'^T} w_\mathcal{S} - \frac{1}{2} w_\mathcal{S}^T \left(C_\mathcal{S}^{\mathcal{A}'} + \lambda I\right) w_\mathcal{S} \right] = \frac{1}{2} b_\mathcal{S}^{\mathcal{A}'^T} \left(C_\mathcal{S}^{\mathcal{A}'} + \lambda I\right)^{-1} b_\mathcal{S}^{\mathcal{A}'}.
$$

As an aside, tying back to the unregularized case [Das and Kempe, 2011], the corresponding result is equivalent to setting $\lambda = 0$, where the b and C matrices are simply the residual

covariance vector and matrix for the set $\mathcal{S}$, with respect to the set $\mathcal{A}$:

$$b_{\mathcal{S}}^{\mathcal{A}} = b_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} C_{\mathcal{A}}^{-1} b_{\mathcal{A}}$$
$$C_{\mathcal{S}}^{\mathcal{A}} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} C_{\mathcal{A}}^{-1} C_{\mathcal{A}\mathcal{S}},$$

so the regularized form subsumes the previous result as one would expect. ■

We can now derive the approximate submodularity bound for the Tikhonov regularized version of the problem, using the above lemmas.

Theorem 5.2.5. *Let variables X_i be zero mean, unit variance random variables and Y be a target variable. Let $\lambda_{min}(C)$ be the minimum eigenvalue of the covariance matrix of the variables X_i. Then F, as given in Equation (5.5) with regularization parameter λ is approximately submodular for all $\gamma \leq \frac{\lambda_{min}(C)+\lambda}{1+\lambda} \leq \frac{\lambda}{1+\lambda}$, and $\delta = 0$.*

Proof. From Definition 4.2.1, we need to show that the following holds:

$$\gamma \left[F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) \right] - \delta \leq \sum_{x \in \mathcal{S}} \left[F(\mathcal{A} \cup \{x\}) - F(\mathcal{A}) \right].$$

By Lemma 5.2.4, the left hand side can be simplified to

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) = \frac{1}{2} {b_{\mathcal{S}}^{\mathcal{A}'}}^T (C_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'}.$$

We can do the same thing to the right hand side and get

$$\begin{aligned} \sum_{x \in \mathcal{S}} F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) &= \sum_{x \in \mathcal{S}} \frac{1}{2} {b_{\{x\}}^{\mathcal{A}}}'^T ({C_{\{x\}}^{\mathcal{A}}}' + \lambda I)^{-1} {b_{\{x\}}^{\mathcal{A}}}' \\ &= \frac{1}{2} {b_{\mathcal{S}}^{\mathcal{A}'}}^T \text{diag}(C_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'}. \end{aligned}$$

We can lower bound the right hand side further using the variance bound on the variables X_i, giving $C_x \leq 1$ and the fact that $C_{x\mathcal{A}} \left(C_{\mathcal{A}} + \lambda I \right)^{-1} C_{\mathcal{A}x} \geq 0$, to find

$$\frac{1}{2} {b_{\mathcal{S}}^{\mathcal{A}'}}^T \text{diag}(C_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'} \geq \frac{1}{2} \frac{{b_{\mathcal{S}}^{\mathcal{A}'}}^T b_{\mathcal{S}}^{\mathcal{A}'}}{1+\lambda}$$

Thus it suffices to find γ and δ such that

$$\gamma \left(\frac{1}{2} b_S^{A'T} (C_S^{A'} + \lambda I)^{-1} b_S^{A'} \right) - \delta \leq \frac{1}{2} \frac{b_S^{A'T} b_S^{A'}}{1+\lambda}.$$

Using Lemma 5.2.3, we know that

$$\lambda_{\min}(C_S^{A'} + \lambda I) \left(b_S^{A'T} (C_S^{A'} + \lambda I)^{-1} b_S^{A'} \right) \leq b_S^{A'T} b_S^{A'},$$

so the bound holds for $\gamma \leq \frac{\lambda_{\min}(C_S^{A'} + \lambda I)}{1+\lambda} = \frac{\lambda_{\min}(C_S^{A'}) + \lambda}{1+\lambda}$ and $\delta = 0$.

Using Lemma 5.2.1 and Lemma 5.2.2, we know that

$$\lambda_{\min}(C_S^{A'}) \geq \lambda_{\min}(C_S^{A}) \geq \lambda_{\min}(C),$$

giving the final bound:

$$\gamma \leq \frac{\lambda_{\min}(C) + \lambda}{1+\lambda} \leq \frac{\lambda}{1+\lambda}.$$

■

This result subsumes the previous result given by Das and Kempe [2011], and is identical for the unregularized, $\lambda = 0$ case. Additionally, it indicates that if the optimal weight vector is small and not substantially affected by strong regularization, the constants in the approximate submodularity bound are stronger than those given in the previous result, especially when the spectral bound $\lambda_{\min}(C)$ is small.

We can also derive a similar bound for the approximation error introduced by using the Orthogonal Matching Pursuit algorithm on this problem.

Theorem 5.2.6. *Let variables X_i be zero mean, unit variance random variables and Y be a target variable. Let $\lambda_{min}(C)$ be the minimum eigenvalue of the covariance matrix of the variables X_i. The OMP algorithm applied to F as given in Equation (5.5) with regularization parameter λ is $(\alpha, 0)$-approximately greedy as given in Definition 4.3.1, with $\alpha = \frac{\lambda_{min(C)} + \lambda}{1+\lambda}$.*

Proof. The OMP algorithm (Algorithm 5.2) at iteration $j+1$ selects the element x^* which maximizes

$$x^* = \arg\max_{x \in \mathcal{X}} \frac{\mathbb{E}[(Y - w^{*T} X_{\mathcal{D}_{j-1}})^T x]^2}{c(x)},$$

where

$$w^* = \arg\min_w \mathbb{E}[(Y - w^T X_{\mathcal{D}_{j-1}})^2].$$

We can compute w^* using the covariance vector b and matrix C directly to be

$$w^* = \left(C_{\mathcal{D}_j} + \lambda I\right)^{-1} b_{\mathcal{D}_j}.$$

So OMP selects the element which maximizes:

$$\begin{aligned}
&\frac{\mathbb{E}[(Y - \left(\left(C_{\mathcal{D}_j} + \lambda I\right)^{-1} b_{\mathcal{D}_j}\right)^T X_{\mathcal{D}_{j-1}})x]^2}{c(x)} \\
&= \frac{\left(b_x - \left(\left(C_{\mathcal{D}_j} + \lambda I\right)^{-1} b_{\mathcal{D}_j}\right)^T \mathbb{E}[X_{\mathcal{D}_{j-1}} x]\right)^2}{c(x)} \\
&= \frac{\left(b_x - \left(\left(C_{\mathcal{D}_j} + \lambda I\right)^{-1} b_{\mathcal{D}_j}\right)^T C_{\mathcal{D}_j x}\right)^2}{c(x)} \\
&= \frac{\left(b_x^{\mathcal{D}_j\prime}\right)^2}{c(x)},
\end{aligned}$$

where

$$b_x^{\mathcal{D}_j\prime} = b_x - C_{x\mathcal{D}_j}\left(C_{\mathcal{D}_j} + \lambda I\right)^{-1} b_{\mathcal{D}_j}.$$

This implies that

$$\frac{b_{x^*}^{\mathcal{D}_j\prime T} b_{x^*}^{\mathcal{D}_j\prime}}{c(x^*)} \geq \max_x \frac{b_x^{\mathcal{D}_j\prime T} b_x^{\mathcal{D}_j\prime}}{c(x)}$$

By Lemma 5.2.4, the gain for a single element x at iteration $j+1$ is

$$F(\mathcal{D}_j \cup \{x\}) - F(\mathcal{D}_j) = \frac{1}{2} b_x^{\mathcal{D}_j\prime T}\left(C_x^{\mathcal{D}_j\prime} + \lambda I\right)^{-1} b_x^{\mathcal{D}_j\prime}.$$

Using Lemma 5.2.1 through Lemma 5.2.3, in a similar argument to the previous proof, we find that, for all x

$$b_x^{\mathcal{D}_j\prime T} b_x^{\mathcal{D}_j\prime} \geq \left(\lambda_{\min}(C) + \lambda\right) b_x^{\mathcal{D}_j\prime T}\left(C_x^{\mathcal{D}_j\prime} + \lambda I\right)^{-1} b_x^{\mathcal{D}_j\prime}.$$

Using this, along with the fact that $C_{x^*}^{\mathcal{D}_j\prime} < 1$, gives

$$\begin{aligned}
\frac{F(\mathcal{D}_j \cup \{x^*\}) - F(\mathcal{D}_j)}{c(x^*)} &= \frac{b_{x^*}^{\mathcal{D}_j\prime T}\left(C_{x^*}^{\mathcal{D}_j\prime} + \lambda I\right)^{-1} b_{x^*}^{\mathcal{D}_j\prime}}{c(x^*)} \\
&\geq \frac{1}{1+\lambda}\frac{b_{x^*}^{\mathcal{D}_j\prime T} b_{x^*}^{\mathcal{D}_j\prime}}{c(x^*)} \\
&\geq \frac{1}{1+\lambda}\max_x \frac{b_x^{\mathcal{D}_j\prime T} b_x^{\mathcal{D}_j\prime}}{c(x)} \\
&\geq \max_x \frac{\lambda_{\min}(C)+\lambda}{1+\lambda}\frac{b_x^{\mathcal{D}_j\prime T}\left(C_x^{\mathcal{D}_j\prime} + \lambda I\right)^{-1} b_x^{\mathcal{D}_j\prime}}{c(x)} \\
&= \max_x \frac{\lambda_{\min}(C)+\lambda}{1+\lambda}\frac{F(\mathcal{D}_j \cup \{x\}) - F(\mathcal{D}_j)}{c(x)}
\end{aligned}$$

completing the proof. ∎

This result also subsumes the previous corresponding OMP result for the unregularized version of the problem [Das and Kempe, 2011]. Combined with the previous theorem and the submodular optimization analyses in Chapter 4 we can now directly derive approximation bounds for the regularized sparse approximation problem.

Corollary 5.2.7. *Let $\gamma = \frac{\lambda_{min(C)}+\lambda}{1+\lambda}$. Let F be the regularized sparse approxmation objective given in Equation (5.5). Let $S = (s_1, \ldots)$ be any sequence. Let $D = (d_1, \ldots)$ be the sequence selected by the greedy Forward Regression algorithm. Fix some $K > 0$. Let $B = \sum_{i=1}^{K} c(d_i)$. Then*

$$F(\mathcal{D}_{\langle B\rangle}) > (1 - e^{-\gamma\frac{B}{C}})F(\mathcal{S}_{\langle C\rangle}).$$

Similarly let $D' = (d'_1, \ldots)$ be the sequence selected by the Orthogonal Matching Pursuit algorithm and $B' = \sum_{i=1}^{K} c(d'_i)$. Then

$$F(\mathcal{D}'_{\langle B'\rangle}) > (1 - e^{-\gamma^2\frac{B'}{C}})F(\mathcal{S}_{\langle C\rangle}).$$

5.3 Constrained Sparse Approximation

An alternative problem to consider that also addresses the concern of large weights is the Ivanov regularized, or constrained variant of the sparse approximation problem. Under this approach we

constrain the weight vectors applied to the selected subset to lie within some ϵ-ball:

$$F(\mathcal{S}) = \frac{1}{2}\mathbb{E}[Y^2] - \min_{w \in \mathbb{R}^{|\mathcal{S}|}, \|w\|_2 \leq \epsilon} \frac{1}{2}\mathbb{E}[(Y - w^T X_{\mathcal{S}})^2] \tag{5.7}$$

$$= \max_{w \in \mathbb{R}^{|\mathcal{S}|}, \|w\|_2 \leq \epsilon} b_{\mathcal{S}}^T w - \frac{1}{2} w^T C_{\mathcal{S}} w. \tag{5.8}$$

This constrained version of the problem allows for approximation gaurantees in terms of a bound directly on the weight vector used in the optimal solution. By directly constraining the allowable weights, we restrict the impact that large weights could have on the approximate submodularity of the problem. Unlike the previous regularization approach, however, a constrained approach does not change the optimal solution to the problem, as long as the constraint is sufficiently large.

We can now detail the same approximate submodularity and approximately greedy bounds for the constrained case. We can first derive a similar result to Lemma 5.2.4 which simplifies the gain $F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})$ for the constrained problem.

Lemma 5.3.1. *For some variables X_i, and Y, let F be defined as in Equation (5.7) with constraint ϵ. Let $w^* = \arg\min_{\|w\|_2 \leq \epsilon} \mathbb{E}[(Y - w^T X_{\mathcal{A}})^2]$ be the weight vector which maximizes $F(\mathcal{A})$. Then*

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) = \max_{\left\|w + w^*_{\mathcal{A} \cup \mathcal{S}}\right\|_2 \leq \epsilon} \left(b_{\mathcal{A} \cup \mathcal{S}} - C_{\mathcal{A} \cup \mathcal{S} w^*_{\mathcal{A} \cup \mathcal{S}}}\right)^T w - \frac{1}{2} w^T C_{\mathcal{A} \cup \mathcal{S}} w,$$

where

$$w^*_{\mathcal{A} \cup \mathcal{S}} = \begin{bmatrix} w^* \\ 0_{\mathcal{S}} \end{bmatrix}.$$

Proof. Starting with the definition of F from Equation (5.7), we have

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) = \max_{\|w\|_2 \leq \epsilon} \left[b^T_{\mathcal{A} \cup \mathcal{S}} w - \frac{1}{2} w^T C_{\mathcal{A} \cup \mathcal{S}} w \right] - \max_v \left[b^T_{\mathcal{A}} v - \frac{1}{2} v^T C_{\mathcal{A}} v \right].$$

Let $w^*_{\mathcal{A} \cup \mathcal{S}}$ be w^* extended to the dimension of $\mathcal{A} \cup \mathcal{S}$ with zeros:

$$w^*_{\mathcal{A} \cup \mathcal{S}} = \begin{bmatrix} w^* \\ 0_{\mathcal{S}} \end{bmatrix}.$$

Re-writing using the definition of w^* gives

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})$$
$$= \max_{\|w\|_2 \leq \epsilon} \left[b_{\mathcal{A}\cup\mathcal{S}}^T (w + w_{\mathcal{A}\cup\mathcal{S}}^*) - \frac{1}{2}(w + w_{\mathcal{A}\cup\mathcal{S}}^*)^T C_{\mathcal{A}\cup\mathcal{S}}(w + w_{\mathcal{A}\cup\mathcal{S}}^*) \right] - b_{\mathcal{A}}^T w^* + \frac{1}{2} w^{*T} C_{\mathcal{A}} w^*.$$

Expanding out terms and cancelling completes the proof:

$$\begin{aligned}
&= \max_{\left\|w+w_{\mathcal{A}\cup\mathcal{S}}^*\right\|_2 \leq \epsilon} \Bigg[b_{\mathcal{A}\cup\mathcal{S}}^T w_{\mathcal{A}\cup\mathcal{S}}^* + b_{\mathcal{A}\cup\mathcal{S}}^T w - \frac{1}{2} w^T C_{\mathcal{A}\cup\mathcal{S}} w - {w_{\mathcal{A}\cup\mathcal{S}}^*}^T C_{\mathcal{A}\cup\mathcal{S}} w \\
&\qquad\qquad - \frac{1}{2} {w_{\mathcal{A}\cup\mathcal{S}}^*}^T C_{\mathcal{A}\cup\mathcal{S}} w_{\mathcal{A}\cup\mathcal{S}}^* \Bigg] - b_{\mathcal{A}}^T w^* + \frac{1}{2} w^{*T} C_{\mathcal{A}} w^* \\
&= \max_{\left\|w+w_{\mathcal{A}\cup\mathcal{S}}^*\right\|_2 \leq \epsilon} \Bigg[b_{\mathcal{A}}^T w^* + b_{\mathcal{A}\cup\mathcal{S}}^T w - \frac{1}{2} w^T C_{\mathcal{A}\cup\mathcal{S}} w - {w_{\mathcal{A}\cup\mathcal{S}}^*}^T C_{\mathcal{A}\cup\mathcal{S}} w \\
&\qquad\qquad - \frac{1}{2} w^{*T} C_{\mathcal{A}} w^* \Bigg] - b_{\mathcal{A}}^T w^* + \frac{1}{2} w^{*T} C_{\mathcal{A}} w^* \\
&= \max_{\left\|w+w_{\mathcal{A}\cup\mathcal{S}}^*\right\|_2 \leq \epsilon} \left[\left(b_{\mathcal{A}\cup\mathcal{S}} - C_{\mathcal{A}\cup\mathcal{S}} w_{\mathcal{A}\cup\mathcal{S}}^*\right)^T w - \frac{1}{2} w^T C_{\mathcal{A}\cup\mathcal{S}} w \right].
\end{aligned}$$

■

Unfortunately in the constrained setting there is no analytic closed-form solution, as there was in the previous case. The resulting expression for the gain is essentially a simplified maximiation problem with a modified constraint.

Now, to derive the desired result we want to eventually bound the term for the combined gain in terms of the individual gains. Unfortunately a complete proof of this bound is still an open problem. We are able to prove the bound for a number of special cases.

The difficulty in getting a complete proof for this problem is the lack of closed form solution when dealing with the constrained version of the problem, in particular when analyzing the gain for a set $\mathcal{S}$, $F(\mathcal{S} \cup \mathcal{A}) - F(\mathcal{A})$. In the analysis of the Tikhonov regularized and unregularized versions of the sparse approximation, the convenient closed form solutions are used to show that this difference of two quadratic optimizations is equivalent to a single quadratic optimization over only the variables in $\mathcal{S}$.

Conjecture 5.3.2. *Let variables X_i be zero mean, unit variance random variables and Y be a target variable, and b and C the appropriate covariance vector and matrix. Let $\mathcal{S}$ and $\mathcal{A}$ be sets*

of the variables X_i. Let $w^* = \arg\min_{\|w\|_2 \leq \epsilon} \mathbb{E}[(Y - w^T X_{\mathcal{A}})^2]$. Then, for all γ

$$\gamma \left(\max_{\left\|w + w^*_{\mathcal{A}\cup\mathcal{S}}\right\|_2 \leq \epsilon} \left(b_{\mathcal{A}\cup\mathcal{S}} - C_{\mathcal{A}\cup\mathcal{S}} w^*_{\mathcal{A}\cup\mathcal{S}}\right)^T w - \frac{1}{2} w^T C_{\mathcal{A}\cup\mathcal{S}} w \right) - \frac{\gamma^2\epsilon^2}{2}$$
$$\leq \sum_{x\in\mathcal{S}} \max_{\left\|w + w^*_{\mathcal{A}\cup\{x\}}\right\|_2 \leq \epsilon} \left(b_{\mathcal{A}\cup\{x\}} - C_{\mathcal{A}\cup\{x\}} w^*_{\mathcal{A}\cup\{x\}}\right)^T w - \frac{1}{2} w^T C_{\mathcal{A}\cup\{x\}} w,$$

where

$$w^*_{\mathcal{A}\cup\mathcal{S}} = \begin{bmatrix} w^* \\ 0_{\mathcal{S}} \end{bmatrix}.$$

Proof. This conjecture is the critical failure point in our current understanding of the constrained sparse approximation problem. We do not currently have a proof of the complete statement, but we can prove it for certain special cases.

Specifically, we now detail a proof of the case when $\mathcal{A} = \emptyset$. From numerical experiments and other exploration of the problem, we believe that this case is the tightest case for the bound.

Eliminating the terms that depend on $\mathcal{A}$, we can reduce the left and right hand side of the problem such that we need to find γ and δ such that:

$$\gamma \left(\max_{\|w\|_2 \leq \epsilon} b^T_{\mathcal{S}} w - \frac{1}{2} w^T C_{\mathcal{S}} w \right) - \delta \leq \sum_{x\in S} \max_{\|w\|_2 \leq \epsilon} b^T_{\{x\}} w - \frac{1}{2} w^T C_{\{x\}} w.$$

Using the fact that $C_{\mathcal{S}}$ is positive semi-definite and Cauchy-Schwarz we can upper bound the left hand side:

$$\gamma \left(\max_{\|w\|_2 \leq \epsilon} b^T_{\mathcal{S}} w - \frac{1}{2} w^T C_{\mathcal{S}} w \right) - \delta \leq \gamma \|b_{\mathcal{S}}\|_2 \epsilon - \delta$$

Similarly we can reduce the right hand side to a single optimization and bound:

$$\sum_{x\in S} \max_{\|w\|_2 \leq \epsilon} b^T_{\{x\}} w - \frac{1}{2} w^T C_{\{x\}} w = \max_{\|w\|_\infty \leq \epsilon} b^T_{\mathcal{S}} w - \frac{1}{2} w^T w$$
$$\geq \max_{\|w\|_2 \leq \epsilon} b^T_{\mathcal{S}} w - \frac{1}{2} w^T w.$$

Thus it suffices to find γ and δ such that

$$\gamma \|b_{\mathcal{S}}\|_2 \epsilon - \delta \leq \max_{\|w\|_2 \leq \epsilon} b^T_{\mathcal{S}} w - \frac{1}{2} w^T w.$$

When $\|b_{\mathcal{S}}\|_2 \geq \epsilon$, the right hand side is maximized at $w = \frac{\epsilon b_{\mathcal{S}}}{\|b_{\mathcal{S}}\|_2}$, and thus we want γ and δ such that

$$\gamma \|b_{\mathcal{S}}\|_2 \epsilon - \delta \leq \|b_{\mathcal{S}}\|_2 \epsilon - \frac{1}{2}\epsilon^2$$

which holds true for any $\gamma \leq 1$ and $\delta \geq (\gamma - \frac{1}{2})\epsilon^2$.

When $\|b_{\mathcal{S}}\|_2 \leq \epsilon$, the right hand side is maximized at $w = b_{\mathcal{S}}$, giving:

$$\gamma \|b_{\mathcal{S}}\|_2 \epsilon - \delta \leq \frac{1}{2}\|b_{\mathcal{S}}\|_2^2$$

which holds for any $\gamma \leq 1$ and $\delta \geq \frac{\gamma^2\epsilon^2}{2}$. Since $\frac{1}{2}\gamma^2 > (\gamma - \frac{1}{2})$ for $\gamma \leq 1$, the second set of constraints satisfies both cases. ■

Using this (conjectured) result, we can derive the corresponding approximate submodularity and approximately greedy bounds. For the remainder of this document, and bounds related to the constrained setting rely on the previous conjecture, and as a result are also conjectured results. We will also note this for each conjectured result.

Theorem 5.3.3 ([a])**.** *Let variables X_i be zero mean, unit variance random variables and Y be a target variable. Then F, as defined in Equation (5.7) with constraint ϵ is approximately submodular for all $\gamma \in [0, 1]$ and $\delta \geq \frac{\gamma^2\epsilon^2}{2}$.*

[a] This theorem requires that Conjecture 5.3.2 hold in general.

Proof. Let $w^* = \arg\min_{\|w\|_2 \leq \epsilon} \mathbb{E}[(Y - w^T X_{\mathcal{A}})^2]$, and

$$w^*_{\mathcal{A}\cup\mathcal{S}} = \begin{bmatrix} w^* \\ 0_{\mathcal{S}} \end{bmatrix}.$$

By Lemma 5.3.1 the left hand side gain can be re-written as

$$F(\mathcal{A}\cup\mathcal{S}) - F(\mathcal{A}) = \max_{\|w+w^*_{\mathcal{A}\cup\mathcal{S}}\|_2 \leq \epsilon} \left(b_{\mathcal{A}\cup\mathcal{S}} - C_{\mathcal{A}\cup\mathcal{S}} w^*_{\mathcal{A}\cup\mathcal{S}}\right)^T w - \frac{1}{2} w^T C_{\mathcal{A}\cup\mathcal{S}} w.$$

Similarly, the right hand side gain can be re-written as

$$\sum_{x\in\mathcal{S}} F(\mathcal{A}\cup\{x\}) - F(\mathcal{A}) = \sum_{x\in\mathcal{S}} \max_{\|w+w^*_{\mathcal{A}\cup x}\|_2 \leq \epsilon} \left(b_{\mathcal{A}\cup\{x\}} - C_{\mathcal{A}\cup\{x\}} w^*_{\mathcal{A}\cup\{x\}}\right)^T w - \frac{1}{2} w^T C_{\mathcal{A}\cup\{x\}} w.$$

By Conjecture 5.3.2, the following holds:

$$\gamma\left(\max_{\left\|w+w^*_{\mathcal{A}\cup\mathcal{S}}\right\|_2\leq\epsilon}\left(b_{\mathcal{A}\cup\mathcal{S}}-C_{\mathcal{A}\cup\mathcal{S}}w^*_{\mathcal{A}\cup\mathcal{S}}\right)^T w-\frac{1}{2}w^T C_{\mathcal{A}\cup\mathcal{S}}w\right)-\frac{\gamma^2\epsilon^2}{2}$$
$$\leq\sum_{x\in\mathcal{S}}\max_{\left\|w+w^*_{\mathcal{A}\cup\{x\}}\right\|_2\leq\epsilon}\left(b_{\mathcal{A}\cup\{x\}}-C_{\mathcal{A}\cup\{x\}}w^*_{\mathcal{A}\cup\{x\}}\right)^T w-\frac{1}{2}w^T C_{\mathcal{A}\cup\{x\}}w,$$

which implies that the theorem holds for $\gamma\in[0,1]$ and $\delta\geq\frac{\gamma^2\epsilon^2}{2}$. ■

In this particular case, the bound holds for any γ, with larger γ improving the multiplicative approximation, but also weakening the additive bound.

The matching approximation error bound for the OMP algorithm applied to the constrained problem is also an open problem, but analysis of special cases leads us to believe that, as in the other sparse approximation problems, the bound matches the approximate submodularity bound.

Theorem 5.3.4 ([a])**.** *Let variables X_i be zero mean, unit variance random variables and Y be a target variable. Then the OMP algorithm appplied to F, as defined in Equation (5.7) with constraint ϵ is (α,β)-approximately greedy as given in Definition 4.3.1, for all $\alpha\in[0,1]$ and $\beta\geq\frac{\alpha^2\epsilon^2}{2}$.*

[a] This theorem is based on Conjecture 5.3.2 ultimately being true.

Proof. Let $w^*=\arg\min_{\|w\|_2\leq\epsilon}\mathbb{E}[(Y-w^TX_{\mathcal{D}_j})^2]$ and

$$w^*_{\mathcal{D}_j\cup\mathcal{S}}=\begin{bmatrix}w^*\\0_{\mathcal{S}}\end{bmatrix}.$$

The OMP algorithm (Algorithm 5.2) at iteration $j+1$ selects the element x^* which maximizes

$$x^*=\arg\max_{x\in\mathcal{X}}\frac{\mathbb{E}[(Y-w^{*T}X_{\mathcal{D}_j})^Tx]^2}{c(x)}.$$

So OMP selects the element which maximizes:

$$\frac{\mathbb{E}[(Y-w^{*T}X_{\mathcal{D}_{j-1}})^Tx]^2}{c(x)}=\frac{\left(b_x-w^{*T}C_{\mathcal{D}_jx}\right)^2}{c(x)}$$
$$=\frac{\left(b_x^{\mathcal{D}_j\prime}\right)^2}{c(x)},$$

where

$$b_x^{\mathcal{D}_j\prime} = b_x - C_{x\mathcal{D}_j} w^*.$$

This implies that

$$\frac{b_{x^*}^{\mathcal{D}_j\prime T} b_{x^*}^{\mathcal{D}_j\prime}}{c(x^*)} \geq \max_x \frac{b_x^{\mathcal{D}_j\prime T} b_x^{\mathcal{D}_j\prime}}{c(x)}$$

By Lemma 5.3.1, the gain for a single element x at iteration $j+1$ is

$$(F(\mathcal{D}_j \cup \{x\}) - F(\mathcal{D})_j) = \max_{\left\| w + w^*_{\mathcal{D}_j \cup \{x\}} \right\|_2 \leq \epsilon} \left(b_{\mathcal{D}_| \cup \{x\}} - C_{\mathcal{D}_j \cup \{x\}} w^*_{\mathcal{D}_j \cup \{x\}} \right)^T w - \frac{1}{2} w^T C_{\mathcal{D}_j \cup \{x\}} w.$$

Note that the portion of $b_{\mathcal{D}_| \cup \{x\}}$ which corresponds to x is exactly the $b_x^{\mathcal{D}_j\prime}$ term maximized by OMP.

We hypothesize that the same argument which could prove Conjecture 5.3.1 should be able to prove the rest of this theorem as well. Namely, we hypothesize that we should be able to bound the gain of the selected element as some function of $b_{x^*}^{\mathcal{D}_j\prime T} b_{x^*}^{\mathcal{D}_j\prime}$. Then, using the OMP maximization criteria we can bound that in terms of $b_x^{\mathcal{D}_j\prime T} b_x^{\mathcal{D}_j\prime}$.

Using a proof of Conjecture 5.3.2, we should be able to show that that function of $b_x^{\mathcal{D}_j\prime T} b_x^{\mathcal{D}_j\prime}$ is bounded by $\gamma(F(\mathcal{D}_j \cup \{x\}) - F(\mathcal{D})_j) - \frac{\gamma^2\epsilon^2}{2}$, completing the proof. ■

Assuming that these conjectures are true gives a set of corresponding approximation bounds for greedy approaches to the constrained sparse approximation problem.

Corollary 5.3.5 ([a]). *Let $\gamma \in [0,1]$. Let F be the constrained sparse approxmation objective given in Equation (5.7) with constraint ϵ. Let $S = (s_1, \ldots)$ be any sequence. Let $D = (d_1, \ldots)$ be the sequence selected by the greedy Forward Regression algorithm. Fix some $K > 0$. Let $B = \sum_{i=1}^{K} c(d_i)$. Then*

$$F(\mathcal{D}_{\langle B \rangle}) > (1 - e^{-\gamma \frac{B}{C}}) F(\mathcal{S}_{\langle C \rangle}) - \frac{\gamma^2\epsilon^2}{2} \frac{B}{C}.$$

Similarly let $D' = (d'_1, \ldots)$ be the sequence selected by the Orthogonal Matching Pursuit algorithm and $B' = \sum_{i=1}^{K} c(d'_i)$. Then

$$F(\mathcal{D}'_{\langle B' \rangle}) > (1 - e^{-\gamma^2 \frac{B'}{C}}) F(\mathcal{S}_{\langle C \rangle}) - \frac{1+\gamma}{2} \gamma^2 \epsilon^2 \frac{B'}{C}.$$

[a] This theorem is based on Conjecture 5.3.2 ultimately being true.

5.4 Generalization to Smooth Losses

The results in previous sections are all for the sparse approximation problem which directly optimizes the squared reconstruction error with respect to some target Y. In many domains we would like to use the same basic subset selection strategy, but with a different loss function to optimize. In this section we extend the previous results to arbitrary smooth losses, using the previous analysis as a starting point.

Recall that a loss function ℓ is m-*strongly convex* if for all x, x':

$$\ell(x') \geq \ell(x) + \langle \nabla \ell(x), x' - x \rangle + \frac{m}{2} \|x' - x\|^2 \tag{5.9}$$

for some $m > 0$, and M-*strongly smooth* if

$$\ell(x') \leq \ell(x) + \langle \nabla \ell(x), x' - x \rangle + \frac{M}{2} \|x' - x\|^2 \tag{5.10}$$

for some $M > 0$.

The sparse approximation for arbitrary losses simply replaces the squared error with a smooth loss ℓ. The equivalent problem to the squared loss problem in coordinate space given in Equation (5.1) is

$$f(w) = \mathbb{E}[\ell(w^T X_{\mathcal{S}})]. \tag{5.11}$$

We can turn this in to a monotonic, positive set function by subtracting that value from the starting loss, giving the equivalent set function for the smooth case:

$$F(\mathcal{S}) = \mathbb{E}[\ell(0)] - \min_{w \in \mathbb{R}^{|\mathcal{S}|}} \mathbb{E}[\ell(w^T X_{\mathcal{S}})]. \tag{5.12}$$

To continue our analysis above, we can also generalize the Tikhonov regularized version of the problem. The function to optimize in coordinate space is

$$f(w) = \mathbb{E}[\ell(w^T X_{\mathcal{S}}) + \frac{\lambda}{2} w^T w], \tag{5.13}$$

and the equivalent set function is

$$F(\mathcal{S}) = \mathbb{E}[\ell(0)] - \min_{w \in \mathbb{R}^{|\mathcal{S}|}} \mathbb{E}[\ell(w^T X_{\mathcal{S}}) + \frac{\lambda}{2} w^T w], \tag{5.14}$$

where the $\mathbb{E}[\ell(0)]$ term is included to transform the problem in to a positive set function maximization.

We can now analyze the approximate submodularity and approximate greedy behavior of the regularized, smooth loss version of the problem. First, we need to develop upper and lower bounds of the gain terms, using the strong smoothness and strong convexity of the loss ℓ. Unlike the squared loss case, we don't get an exact expression for the gain terms, only quadratic upper and lower bounds.

We first give the lower bound for the gain terms, utilizing the strong smoothness of the loss.

Lemma 5.4.1. *Let F be as given in Equation (5.14) for ℓ, an M-strongly smooth loss as in Equation (5.10). Let $w^* = \arg\min_w \mathbb{E}[\ell(w^T X_{\mathcal{A}}) + \frac{\lambda}{2} w^T w]$ be the weight vector which maximizes $F(\mathcal{A})$, and $Z = w^* X_{\mathcal{A}}$. Then*

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) \geq \frac{1}{2} b_{\mathcal{S}}^{\mathcal{A}'^T} (M C_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'},$$

where

$$b_{\mathcal{S}}^{\mathcal{A}'} = -\mathbb{E}[\nabla \ell(Z) X_{\mathcal{S}}]$$
$$C_{\mathcal{S}}^{\mathcal{A}'} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \frac{\lambda}{M} I \right)^{-1} C_{\mathcal{A}\mathcal{S}}$$

Proof. Starting with the definition of F from Equation (5.14), we have

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) = \min_w \mathbb{E}[\ell(w^T X_{\mathcal{A}}) + \frac{\lambda}{2} w^T w] - \min_w \mathbb{E}[\ell(w^T X_{\mathcal{A}\cup\mathcal{S}}) + \frac{\lambda}{2} w^T w].$$

Let $w^*_{\mathcal{A}\cup\mathcal{S}}$ be w^* extended to the dimension of $\mathcal{A} \cup \mathcal{S}$ with zeros:

$$w^*_{\mathcal{A}\cup\mathcal{S}} = \begin{bmatrix} w^* \\ 0_{\mathcal{S}} \end{bmatrix}.$$

Using w^* and Z, and expanding using the strong smoothness requirement around Z:

$$\begin{aligned}
&F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) \\
&= \mathbb{E}[\ell(Z) + \frac{\lambda}{2} w^{*T} w^*] - \min_w \mathbb{E}[\ell(Z + w^T X_{\mathcal{A}\cup\mathcal{S}}) + \frac{\lambda}{2}(w + w_{\mathcal{A}\cup\mathcal{S}})^T (w + w_{\mathcal{A}\cup\mathcal{S}})] \\
&\geq \mathbb{E}[\ell(Z) + \frac{\lambda}{2} w^{*T} w^*] - \min_w \mathbb{E}[\ell(Z) + \left\langle \nabla \ell(Z), w^T X_{\mathcal{A}\cup\mathcal{S}} \right\rangle + \frac{M}{2} \left\| w^T X_{\mathcal{A}\cup\mathcal{S}} \right\|^2 \\
&\qquad + \frac{\lambda}{2}(w + w^*_{\mathcal{A}\cup\mathcal{S}})^T (w + w^*_{\mathcal{A}\cup\mathcal{S}})] \\
&= \max_w \mathbb{E}[-\left\langle \nabla \ell(Z), w^T X_{\mathcal{A}\cup\mathcal{S}} \right\rangle - \frac{M}{2} \left\| w^T X_{\mathcal{A}\cup\mathcal{S}} \right\|^2 - \frac{\lambda}{2} w^T w - \lambda w^T w^*_{\mathcal{A}\cup\mathcal{S}}] \\
&= \max_w \mathbb{E}[(-\nabla \ell(Z) X_{\mathcal{A}\cup\mathcal{S}} - \lambda w^*_{\mathcal{A}\cup\mathcal{S}})^T w] - \frac{1}{2} w^T (M C_{\mathcal{A}\cup\mathcal{S}} + \lambda I) w
\end{aligned}$$

Let $b'_{\mathcal{A}\cup\mathcal{S}} = \mathbb{E}[-\nabla \ell(Z) X_{\mathcal{A}\cup\mathcal{S}} - \lambda w^*_{\mathcal{A}\cup\mathcal{S}}]$. The astute reader will notice that this is effectively the negative gradient of the coordinate loss given in Equation (5.13) evaluated at $w^*_{\mathcal{A}\cup\mathcal{S}}$:

$$b'_{\mathcal{A}\cup\mathcal{S}} = -\nabla_{w^*_{\mathcal{A}\cup\mathcal{S}}} f(w^*_{\mathcal{A}\cup\mathcal{S}}),$$

or the gradient in weight-space of the function we are trying to maximize, at the current solution.

By definition of w^*, this is must be some vector:

$$b'_{\mathcal{A}\cup\mathcal{S}} = \begin{bmatrix} 0_{\mathcal{A}} \\ b_{\mathcal{S}}^{\mathcal{A}'} \end{bmatrix},$$

which is 0 across all dimensions corresponding to $\mathcal{A}$.

Because $b'_{\mathcal{A}\cup\mathcal{S}}$ is zero for all dimensions of $\mathcal{A}$, we can simplify the gain by solving directly and using the formula for block matrix inversion on $MC_{\mathcal{A}\cup\mathcal{S}} + \lambda I$:

$$\begin{aligned} &F(\mathcal{A}\cup\mathcal{S}) - F(\mathcal{A}) \\ &\geq \frac{1}{2} {b'_{\mathcal{A}\cup\mathcal{S}}}^T (MC_{\mathcal{A}\cup\mathcal{S}} + \lambda I)^{-1} b'_{\mathcal{A}\cup\mathcal{S}} \\ &= \frac{1}{2} {b_{\mathcal{S}}^{\mathcal{A}'}}^T (MC_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'}, \end{aligned}$$

where

$$C_{\mathcal{S}}^{\mathcal{A}'} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}}(C_{\mathcal{A}} + \frac{\lambda}{M} I)^{-1} C_{\mathcal{A}\mathcal{S}}$$

■

A similar argument can be used to show the corresponding strong convexity bound. We will omit the proof here because it is largely identical, except for the use of the strong convexity lower bound instead of strong smoothness upper bound on the loss ℓ.

Lemma 5.4.2. *Let F be as given in Equation (5.14) for ℓ, an m-strongly convex loss as in Equation (5.10). Let $w^* = \arg\min_w \mathbb{E}[\ell(w^T X_{\mathcal{A}}) + \frac{\lambda}{2} w^T w]$ be the weight vector which maximizes $F(\mathcal{A})$, and $Z = w^* X_{\mathcal{A}}$. Then*

$$F(\mathcal{A}\cup\mathcal{S}) - F(\mathcal{A}) \leq \frac{1}{2} {b_{\mathcal{S}}^{\mathcal{A}'}}^T (mC_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'},$$

where

$$\begin{aligned} b_{\mathcal{S}}^{\mathcal{A}'} &= -\mathbb{E}[\nabla\ell(Z) X_{\mathcal{S}}] \\ C_{\mathcal{S}}^{\mathcal{A}'} &= C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \frac{\lambda}{m} I \right)^{-1} C_{\mathcal{A}\mathcal{S}} \end{aligned}$$

■

Using these bounds, we can bound the approximate submodularity of the smooth sparse approximation problem.

Theorem 5.4.3. *Let variables X_i be zero mean, unit variance random variables, and ℓ be an m-strongly convex and M-strongly smooth loss function as given in Equations (5.9-5.10). Let $\lambda_{min}(C)$ be the minimum eigenvalue of the covariance matrix of the variables X_i. Then F, as given in Equation (5.14) with regularization parameter λ is approximately submodular for all $\gamma \leq \frac{m\lambda_{min}(C)+\lambda}{M+\lambda} \leq \frac{\lambda}{M+\lambda}$, and $\delta = 0$.*

Proof. Recall that for approximate submodularity to hold we need to show that there exists γ, δ such that

$$\gamma \left(F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})\right) - \delta \leq \left(F(\mathcal{A} \cup \{x\}) - F(\mathcal{A})\right).$$

Let $b_{\mathcal{S}}^{\mathcal{A}'} = -\mathbb{E}[\nabla \ell(Z) X_{\mathcal{S}}]$. Using Lemma 5.4.1

$$F(\mathcal{A} \cup \{x\}) - F(\mathcal{A}) \geq \frac{1}{2} b_x^{\mathcal{A}'^T} (M C_x^{\mathcal{A}'} + \lambda I)^{-1} b_x^{\mathcal{A}'},$$

where

$$C_x^{\mathcal{A}'} = C_x - C_{x\mathcal{A}} \left(C_{\mathcal{A}} + \frac{\lambda}{m} I\right)^{-1} C_{\mathcal{A}x}.$$

Similarly to the proof of Theorem 5.2.5, using $C_x = 1$ and $C_{x\mathcal{A}} \left(C_{\mathcal{A}} + \frac{\lambda}{m} I\right)^{-1} C_{\mathcal{A}x} \geq 0$, we can show that

$$b_x^{\mathcal{A}'^T} (M C_x^{\mathcal{A}'} + \lambda I)^{-1} b_x^{\mathcal{A}'} \geq \frac{b_x^{\mathcal{A}'^T} b_x^{\mathcal{A}'}}{M + \lambda}.$$

Considering the sum over $\mathcal{S}$ we have

$$\sum_{x \in \mathcal{S}} F(\mathcal{A} \cup \{x\}) - F(\mathcal{A}) \geq \frac{b_{\mathcal{S}}^{\mathcal{A}'^T} b_{\mathcal{S}}^{\mathcal{A}'}}{M + \lambda}$$

for a bound on the right-hand side.

For the combined gain we have:

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) \leq \frac{1}{2} b_{\mathcal{S}}^{\mathcal{A}'^T} (m C_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'},$$

where

$$C_{\mathcal{S}}^{\mathcal{A}'} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \frac{\lambda}{m} I\right)^{-1} C_{\mathcal{A}\mathcal{S}}.$$

By Lemma 5.2.3,

$$b_{\mathcal{S}}^{\mathcal{A}'T}(mC_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1}b_{\mathcal{S}}^{\mathcal{A}'} \leq \frac{b_{\mathcal{S}}^{\mathcal{A}'T}b_{\mathcal{S}}^{\mathcal{A}'}}{\lambda_{\min}(mC_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)}.$$

We can now bound the $\lambda_{\min}$ term:

$$\lambda_{\min}(mC_{\mathcal{S}}^{\mathcal{A}'} + \lambda I) = m\lambda_{\min}(C_{\mathcal{S}}^{\mathcal{A}'}) + \lambda \geq m\lambda_{\min}(C_{\mathcal{S}}^{\mathcal{A}}) + \lambda \geq m\lambda_{\min}(C) + \lambda$$

using the definition of $C_{\mathcal{S}}^{\mathcal{A}'}$ in the second step, Lemma 5.2.1 in the third step and Lemma 5.2.2 in the final step.

So now, setting $\gamma = \frac{m\lambda_{\min}(C)+\lambda}{M+\lambda}$ and $\delta = 0$ we have

$$\begin{aligned}
&\frac{m\lambda_{\min}(C) + \lambda}{M + \lambda}\left(F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})\right) \\
&\leq \frac{1}{2}\frac{m\lambda_{\min}(C) + \lambda}{M + \lambda}b_{\mathcal{S}}^{\mathcal{A}'T}(mC_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1}b_{\mathcal{S}}^{\mathcal{A}'} \\
&\leq \frac{1}{2}\frac{b_{\mathcal{S}}^{\mathcal{A}'T}b_{\mathcal{S}}^{\mathcal{A}'}}{M + \lambda} \\
&\leq \sum_{x\in\mathcal{S}}\left(F(\mathcal{A} \cup \{x\}) - F(\mathcal{A})\right),
\end{aligned}$$

completing the proof. ∎

This result extends the squared loss case to all losses bounded by quadratics. One thing to note is that, in the case where $\lambda_{\min}(C)$ is small, the $m\lambda_{\min}(C)$ term contributes negligibly to the bound, and so we can drop the requirement that ℓ is strongly convex. Instead, in this case we only require that ℓ is convex (or equivalently, that ℓ is 0-strongly convex).

These bounds also similarly extend to the OMP algorithm applied to the smooth sparse approximation setting. In the smooth loss setting, the only change to the OMP algorithm given in Algorithm 5.2 is the gradient-based selection step. In the squared loss case, we implicitly stated the gradient for the squared loss in the selection criteria. Given a currently selected set $\mathcal{D}_{j-1}$, the smooth loss equivalent of the OMP algorithm is to first find w^* using

$$w^* = \arg\min_w \mathbb{E}[\ell(w^T X_{\mathcal{D}_{j-1}})].$$

The next element is then selected using

$$x^* = \arg\max_{x\in\mathcal{X}} \frac{\mathbb{E}[-\nabla\ell(w^{*T}X_{\mathcal{D}_{j-1}})^T x]^2}{c(x)},$$

where $\nabla\ell$ is the gradient of the chosen smooth loss function.

We now extend the approximately greedy bound to this setting, getting the same multiplicative constant as the one derived in the approximately submodularity bound, as expected.

Theorem 5.4.4. *Let variables X_i be zero mean, and ℓ be an m-strongly convex and M-strongly smooth loss function as given in Equations (5.9-5.10). Let $\lambda_{min}(C)$ be the minimum eigenvalue of the covariance matrix of the variables X_i. Then the OMP algorithm applied to F as given in Equation (5.14) with regularization parameter λ is approximately greedy for $\alpha \leq \frac{m\lambda_{min}(C)+\lambda}{M+\lambda} \leq \frac{\lambda}{M+\lambda}$, and $\beta = 0$.*

Proof. Like previous cases, the proof is similar to the proof of approximate submodularity in Theorem 5.4.3.

Let $w^* = \arg\min_w \mathbb{E}[\ell(w^T X_{\mathcal{D}_j}) + \frac{\lambda}{2} w^T w]$ be the weight vector which maximizes $F(\mathcal{D}_j)$, and $Z = w^* X_{\mathcal{D}_j}$.

Let $b_{\mathcal{S}}^{\mathcal{D}_j\prime} = -\mathbb{E}[\nabla\ell(Z) X_{\mathcal{S}}]$. The OMP algorithm (Algorithm 5.2) at iteration $j+1$ selects the element x^* which maximizes

$$x^* = \arg\max_{x \in \mathcal{X}} \frac{\mathbb{E}[-\nabla\ell(Z)x]^2}{c(x)},$$

which implies that

$$\frac{b_{x^*}^{\mathcal{D}_j\prime T} b_{x^*}^{\mathcal{D}_j\prime}}{c(x^*)} \geq \max_x \frac{b_x^{\mathcal{D}_j\prime T} b_x^{\mathcal{D}_j\prime}}{c(x)}$$

Now, to lower bound the gain of the element x^*, using Lemma 5.4.1 and the same technique from the proof of Theorem 5.4.3:

$$F(\mathcal{A} \cup \{x^*\}) - F(\mathcal{A}) \geq \frac{1}{2} \frac{b_{\{x^*\}}^{\mathcal{A}\prime T} b_{\{x^*\}}^{\mathcal{A}\prime}}{M+\lambda}.$$

We can similarly upper bound the maximum gain:

$$\max_x F(\mathcal{A} \cup \{x\}) - F(\mathcal{A}) \leq \frac{1}{2} b_x^{\mathcal{A}\prime T} (m C_x^{\mathcal{A}\prime} + \lambda I)^{-1} b_x^{\mathcal{A}\prime},$$

where

$$C_{\mathcal{S}}^{\mathcal{A}\prime} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \frac{\lambda}{m} I \right)^{-1} C_{\mathcal{A}\mathcal{S}}.$$

Using the same eigenvalue bounds as the previous proof:

$$b_{\mathcal{S}}^{\mathcal{A}'T}(mC_{\mathcal{S}}^{\mathcal{A}'} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}'} \leq \frac{b_{\mathcal{S}}^{\mathcal{A}'T} b_{\mathcal{S}}^{\mathcal{A}'}}{m\lambda_{\min}(C) + \lambda}.$$

So now, setting $\alpha = \frac{m\lambda_{\min}(C)+\lambda}{M+\lambda}$ and $\beta = 0$ we have

$$\begin{aligned}
&\frac{F(\mathcal{D}_{j-1} \cup \{x^*\}) - F(\mathcal{D}_{|-\infty})}{c^*} \\
&\geq \frac{1}{2}\frac{1}{M+\lambda}\frac{b_{\{x^*\}}^{\mathcal{A}'T} b_{\{x^*\}}^{\mathcal{A}'}}{c(x^*)} \\
&\geq \max_x \frac{1}{2}\frac{1}{M+\lambda}\frac{b_x^{\mathcal{A}'T} b_x^{\mathcal{A}'}}{c(x)} \\
&\geq \max_x \frac{m\lambda_{\min}(C)+\lambda}{M+\lambda}\frac{F(\mathcal{A}\cup\{x\}) - F(\mathcal{A})}{c(x)},
\end{aligned}$$

completing the proof. ■

Just as in the previous approximate submodularity proof, we can drop the strong convexity requirement and still obtain an approximately greedy bound.

The previous two theorems give the following overall approximation results, when combined with the analysis in Chapter 4.

Corollary 5.4.5. *Let ℓ be an m-strongly convex and M-strongly smooth loss function. Let $\gamma = \frac{m\lambda_{min(C)}+\lambda}{M+\lambda}$. Let F be the regularized, smooth sparse approxmation objective given in Equation (5.14). Let $S = (s_1, \ldots)$ be any sequence. Let $D = (d_1, \ldots)$ be the sequence selected by the greedy Forward Regression algorithm. Fix some $K > 0$. Let $B = \sum_{i=1}^{K} c(d_i)$. Then*

$$F(\mathcal{D}_{\langle B\rangle}) > (1 - e^{-\gamma\frac{B}{C}})F(\mathcal{S}_{\langle C\rangle}).$$

Similarly let $D' = (d'_1, \ldots)$ be the sequence selected by the Orthogonal Matching Pursuit algorithm and $B' = \sum_{i=1}^{K} c(d'_i)$. Then

$$F(\mathcal{D}'_{\langle B'\rangle}) > (1 - e^{-\gamma^2\frac{B'}{C}})F(\mathcal{S}_{\langle C\rangle}).$$

We can also analyze the same smooth loss version of the constrained, or Ivanov regularized approach.

The smooth, constrained version of the problem, as a set function, is

$$F(\mathcal{S}) = \mathbb{E}[\ell(0)] - \min_{w\in\mathbb{R}^{|\mathcal{S}|}, \|w\|_2 \leq \epsilon} \mathbb{E}[\ell(w^T X_{\mathcal{S}})]. \tag{5.15}$$

We also hypothesize that we should be able to derive similar bounds for the smooth, constrained case as in the constrained case for squared loss.

We can derive the corresponding lemmas that upper and lower bound the gain, as in the regularized smooth loss case.

Lemma 5.4.6. *Let F be as given in Equation (5.15) for ℓ, an M-strongly smooth loss as in Equation (5.10). Let $w^* = \arg\min_{\|w\|_2 \leq \epsilon} \mathbb{E}[\ell(w^T X_{\mathcal{A}})]$ be the weight vector which maximizes $F(\mathcal{A})$, and $Z = w^* X_{\mathcal{A}}$. Then*

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) \geq \max_{\left\|w + w^*_{\mathcal{A} \cup \mathcal{S}}\right\|_2 \leq \epsilon} b'_{\mathcal{S} \cup \mathcal{A}} - \frac{M}{2} w^T C_{\mathcal{A} \cup \mathcal{S}} w$$

where

$$b_{\mathcal{S} \cup \mathcal{A}}{}' = -\mathbb{E}[(\nabla \ell(Z) X_{\mathcal{A} \cup \mathcal{S}})]$$

Proof. Starting with the definition of F from Equation (5.15), we have

$$F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) = \min_{\|w\|_2 \leq \epsilon} \mathbb{E}[\ell(w^T X_{\mathcal{A}})] - \min_{\|w\|_2 \leq \epsilon} \mathbb{E}[\ell(w^T X_{\mathcal{A} \cup \mathcal{S}})].$$

Let $w^*_{\mathcal{A} \cup \mathcal{S}}$ be w^* extended to the dimension of $\mathcal{A} \cup \mathcal{S}$ with zeros:

$$w^*_{\mathcal{A} \cup \mathcal{S}} = \begin{bmatrix} w^* \\ 0_{\mathcal{S}} \end{bmatrix}.$$

Using w^* and Z, and expanding using the strong smoothness requirement around Z:

$$\begin{aligned}
&F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A}) \\
&= \mathbb{E}[\ell(Z)] - \min_{\left\|w + w^*_{\mathcal{A} \cup \mathcal{S}}\right\|_2 \leq \epsilon} \mathbb{E}[\ell(Z + w^T X_{\mathcal{A} \cup \mathcal{S}})] \\
&\geq \mathbb{E}[\ell(Z)] - \min_{\left\|w + w^*_{\mathcal{A} \cup \mathcal{S}}\right\|_2 \leq \epsilon} \mathbb{E}[\ell(Z) + \left\langle \nabla \ell(Z), w^T X_{\mathcal{A} \cup \mathcal{S}} \right\rangle + \frac{M}{2} \left\| w^T X_{\mathcal{A} \cup \mathcal{S}} \right\|^2] \\
&= \max_{\left\|w + w^*_{\mathcal{A} \cup \mathcal{S}}\right\|_2 \leq \epsilon} \mathbb{E}[-\left\langle \nabla \ell(Z), w^T X_{\mathcal{A} \cup \mathcal{S}} \right\rangle - \frac{M}{2} \left\| w^T X_{\mathcal{A} \cup \mathcal{S}} \right\|^2] \\
&= \max_{\left\|w + w^*_{\mathcal{A} \cup \mathcal{S}}\right\|_2 \leq \epsilon} \mathbb{E}[(-\nabla \ell(Z) X_{\mathcal{A} \cup \mathcal{S}})] - \frac{M}{2} w^T C_{\mathcal{A} \cup \mathcal{S}} w
\end{aligned}$$

Let $b'_{\mathcal{A} \cup \mathcal{S}} = \mathbb{E}[-\nabla \ell(Z) X_{\mathcal{A} \cup \mathcal{S}}]$.
This completes the proof.

Just as in the regularized case, this is effectively the negative gradient of the coordinate loss given in Equation (5.11) evaluated at the current optimal weight vector $w^*_{\mathcal{A}\cup\mathcal{S}}$:

$$b'_{\mathcal{A}\cup\mathcal{S}} = -\nabla_{w^*_{\mathcal{A}\cup\mathcal{S}}} f(w^*_{\mathcal{A}\cup\mathcal{S}}),$$

or the gradient in weight-space of the function we are trying to maximize, at the current solution. ■

└

Using the same argument, but expanding using the definition of strong convexity yields the corresponding upper bound.

Lemma 5.4.7. *Let F be as given in Equation (5.15) for ℓ, an m-strongly convex loss as in Equation (5.9). Let $w^* = \arg\min_{\|w\|_2\leq\epsilon} \mathbb{E}[\ell(w^T X_{\mathcal{A}})]$ be the weight vector which maximizes $F(\mathcal{A})$, and $Z = w^* X_{\mathcal{A}}$. Then*

$$F(\mathcal{A}\cup\mathcal{S}) - F(\mathcal{A}) \leq \max_{\left\|w+w^*_{\mathcal{A}\cup\mathcal{S}}\right\|_2\leq\epsilon} b'_{\mathcal{S}\cup\mathcal{A}} - \frac{m}{2} w^T C_{\mathcal{A}\cup\mathcal{S}} w$$

where

$$b_{\mathcal{S}\cup\mathcal{A}}{}' = -\mathbb{E}[(\nabla\ell(Z) X_{\mathcal{A}\cup\mathcal{S}})]$$

■

Using these bounds, and the conjectures in Section 5.3, we also hypothesize that we should be able to derive corresponding approximate submodularity and approximately greedy bounds for the smooth, constrained case. We will not present proofs here, but just state the conjectured results. The proofs for the case when the exsisting set $\mathcal{A} = \emptyset$ are relatively straightfoward, as in the conjectured proof of Conjecture 5.3.1.

Theorem 5.4.8 ([a])**.** *Let variables X_i be zero mean, unit variance random variables, and ℓ be an M-strongly smooth loss function as given in Equation (5.10), with $M \geq 1$. Then F, as given in Equation (5.15) with constraint ϵ is approximately submodular for all $\gamma \leq 1$ and all $\delta \geq \frac{M\gamma^2\epsilon^2}{2}$.*

[a] This theorem requires that Conjecture 5.3.2 hold in general.

Theorem 5.4.9 ([a])**.** *Let variables X_i be zero mean, unit variance random variables, and ℓ an M-strongly smooth loss function as given in Equation (5.10), with $M \geq 1$. Then the OMP algorithm applied to F as given in Equation (5.15) with constraint ϵ is approximately greedy for*

$\alpha \leq 1$ *and all* $\beta \geq \frac{M\gamma^2\epsilon^2}{2}$.

[a]This theorem requires that Conjecture 5.3.2 hold in general.

We can now combine these (conjectured) results with the corresponding greedy optimizations from the previous bound and achieve the following (conjectured) approximation bounds.

Corollary 5.4.10 ([a])**.** *Let ℓ be an M-strongly smooth loss function. Let $\gamma \in [0, 1]$. Let F be the constrained, smooth sparse approxmation objective given in Equation (5.15) with constraint ϵ. Let $S = (s_1, \ldots)$ be any sequence. Let $D = (d_1, \ldots)$ be the sequence selected by the greedy Forward Regression algorithm. Fix some $K > 0$. Let $B = \sum_{i=1}^{K} c(d_i)$. Then*

$$F(\mathcal{D}_{\langle B \rangle}) > (1 - e^{-\gamma \frac{B}{C}})F(\mathcal{S}_{\langle C \rangle}) - \frac{M\gamma^2\epsilon^2}{2}\frac{B}{C}.$$

Similarly let $D' = (d'_1, \ldots)$ be the sequence selected by the Orthogonal Matching Pursuit algorithm and $B' = \sum_{i=1}^{K} c(d'_i)$. Then

$$F(\mathcal{D}'_{\langle B' \rangle}) > (1 - e^{-\gamma^2 \frac{B'}{C}})F(\mathcal{S}_{\langle C \rangle}) - \frac{1+\gamma}{2}M\gamma^2\epsilon^2\frac{B'}{C}.$$

[a]This theorem requires that Conjecture 5.3.2 hold in general.

5.5 Simultaneous Sparse Approximation

We will now examine a common extension of the sparse approximation problem, the *simultaneous sparse approximation* problem. In this problem we want to select a subset of the variables X_i that best reconstruct multiple target signals Y_k simultaneously or optimize multiple smooth losses simultaneously. In this setting the same set of variables is selected to reconstruct all signals, but the linear combination of the variables selected is allowed to vary arbitrarily for each signal. More formally, given some set of problems F_k, the objective for this problem is just the sum of these set functions. For example, in the smooth, regularized case from Equation (5.14), the simultaneous version of the loss is

$$F(\mathcal{S}) = \sum_k F_k(\mathcal{S}) \tag{5.16}$$

$$= \sum_k \mathbb{E}[\ell_k(0)] - \min_{w \in \mathbb{R}^{|\mathcal{S}|}} \mathbb{E}[\ell_k(w^T X_{\mathcal{S}}) + \frac{\lambda_k}{2} w^T w]. \tag{5.17}$$

One example of a problem that fits in this setting is the multiclass setting where a one-vs-all approach is used in combination with a smooth loss. Each of the resulting smooth loss problems corresponds to one of the set function F_k. Other examples include reconstructing multiple targets

Y_k in multi-output regression, or other settings where we want to select the same subset of features to simultaneously minimize multiple loss functions.

The corresponding Forward Regression algorithm for the simultaneous setting is very straightforward. We simply select the element that maximizes the per-unit-cost gain of the complete objective:

$$\frac{F(\mathcal{A} \cup \{x\}) - F(\mathcal{A})}{c(x)}.$$

This is equivalent to just summing the gains for each individual sparse approximation problem and then dividing by the cost of the element. It is relatively easy to show that this simultenous Forward Regression algorithm has the same guarantees as in the individual setting.

Theorem 5.5.1. *For $k = (1, \ldots, K)$, let $F_k(\mathcal{S})$ be (γ_k, δ_k)-approximately submodular. Let $F(\mathcal{S}) = \sum_k F_k(\mathcal{S})$. Let $\gamma \leq \gamma_k$ for all k and $\delta = \sum_{k=1}^{K} \delta_k$. Then $F((S))$ is (γ, δ)-approximately submodular.*

Proof. By the definition of approximate submodularity we know that, for $k = (1, \ldots, K)$

$$\gamma_k \left(F_k(\mathcal{A} \cup \mathcal{S}) - F_k(\mathcal{A})\right) - \delta_k \leq \sum_{x \in \mathcal{S}} F_k(\mathcal{A} \cup \{x\}) - F_k(\mathcal{A}).$$

Summing over k we have that

$$\sum_{k=1}^{K} \gamma_k \left(F_k(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})\right) - \delta_k \leq \sum_{k=1}^{K} \sum_{x \in \mathcal{S}} F_k(\mathcal{A} \cup \{x\}) - F(\mathcal{A}).$$

Now using $\gamma \leq \gamma_k$ and $\delta = \sum_{k=1}^{K} \delta_k$ we have

$$\begin{aligned}
&\gamma \left(F(\mathcal{A} \cup \mathcal{S}) - F(\mathcal{A})\right) - \delta \\
&= \gamma \left(\sum_{k=1}^{K} \gamma_k F_k(\mathcal{A} \cup \mathcal{S}) - F_k(\mathcal{A}) \right) - \delta \\
&\leq \sum_{k=1}^{K} \gamma_k \left(F_k(\mathcal{A} \cup \mathcal{S}) - F_k(\mathcal{A})\right) - \delta_k \\
&\leq \sum_{k=1}^{K} \sum_{x \in \mathcal{S}} F_k(\mathcal{A} \cup \{x\}) - F_k(\mathcal{A}) \\
&= \sum_{x \in \mathcal{S}} F(\mathcal{A} \cup \{x\}) - F(\mathcal{A}).
\end{aligned}$$

Algorithm 5.3 Simultaneous Orthogonal Matching Pursuit

Given: elements $\mathcal{X}$, target $\mathcal{Y}$
Define F as in Equation (5.3)
Let $\mathcal{D}_0 = \emptyset$.
for $j = 1, \ldots$ **do**
  for all k **do**
    Let $w_k^* = \arg\min_w \mathbb{E}[(Y_k - w^T X_{\mathcal{D}_{j-1}})^2]$
  end for
  Let $x^* = \arg\max_{x \in \mathcal{X}} \sum_k \frac{\mathbb{E}[(Y_k - w_k^{*T} X_{\mathcal{D}_{j-1}})^T x]^2}{c(x)}$.
  Let $\mathcal{D}_j = \mathcal{D}_{j-1} \cup \{x^*\}$.
end for

■

This result shows that, given a simultaneous sparse approximation problem, it is approximately submodular with the same multiplicative constant as the loosest one of each of the corresponding individual sparse approximation problems, and the sum of the additive constants.

The adaptation of the OMP algorithm is also intuitive. Whereas before the OMP algorithm selected the element which maximized the squared gradient

$$x^* = \arg\max_{x \in \mathcal{X}} \frac{\mathbb{E}[\nabla^T x]^2}{c(x)},$$

we now select the element which maximizes the sum of squared gradients with respect to each individual sparse approximation problem

$$x^* = \arg\max_{x \in \mathcal{X}} \sum_k \frac{\mathbb{E}[\nabla_k^T x]^2}{c(x)}.$$

A complete version of the Simultaneous OMP [Cotter et al., 2005, Chen and Huo, 2006] algorithm is given in Algorithm 5.3, for the squared loss sparse approximation problem. The equivalent selection criteria for the smooth loss case is

$$x^* = \arg\max_{x \in \mathcal{X}} \sum_k \frac{\mathbb{E}[-\nabla \ell_k (w_k^{*T} X_{\mathcal{D}_{j-1}})^T x]^2}{c(x)}.$$

In a manner similiar to the OMP approximation results previously shown, we can derive similar bounds for the SOMP algorithm applied to those settings. The analysis is not as straightforward as the analysis of the Forward Regression approach, and must be done individually for each different sparse approximation setting. We present here the smooth, regularized version of the proof.

Theorem 5.5.2. *For $k = (1, \ldots, K)$, let $F_k(\mathcal{S})$ be the regularized sparse approximation problem for an m-strongly convex and M-strongly smooth loss ℓ_k, and regularization parameter λ as in Equation (5.14). Let $F(\mathcal{S}) = \sum_{k=1}^{K} F_k(\mathcal{S})$. Then the SOMP algorithm applied to $F((S))$ is $(\alpha, 0)$-approximately greedy, for $\alpha = \frac{m\lambda_{min}(C)+\lambda}{M+\lambda}$.*

Proof. This proof very similar to the proof for regular OMP applied to the regularized, smooth-loss version of the sparse approximation problem in Theorem 5.4.4.

Let $w_k^* = \arg\min_w \mathbb{E}[\ell_k(w^T X_{\mathcal{D}_j}) + \frac{\lambda}{2} w^T w]$ be the weight vector which maximizes $F(\mathcal{D}_j)$, and $Z_k = w_k^* X_{\mathcal{D}_j}$.

Let $b_{\mathcal{S}}^{\mathcal{D}_j\prime}{}_k = -\mathbb{E}[\nabla \ell_k(Z_k) X_{\mathcal{S}}]$. The OMP algorithm (Algorithm 5.2) at iteration $j+1$ selects the element x^* which maximizes

$$x^* = \arg\max_{x \in \mathcal{X}} \sum_{k=1}^{K} \frac{\mathbb{E}[\nabla \ell_k(Z_k) x]^2}{c(x)},$$

which implies that

$$\sum_{k=1}^{K} \frac{b_{x^*}^{\mathcal{D}_j\prime}{}_k^T b_{x^*}^{\mathcal{D}_j\prime}{}_k}{c(x^*)} \geq \max_x \sum_{k=1}^{K} \frac{b_x^{\mathcal{D}_j\prime}{}_k^T b_x^{\mathcal{D}_j\prime}{}_k}{c(x)}$$

Now, to lower bound the gain of the element x^*, using Lemma 5.4.1 and the same technique from the proof of Theorem 5.4.4:

$$\sum_{k=1}^{K} F_k(\mathcal{A} \cup \{x^*\}) - F_k(\mathcal{A}) \geq \sum_{k=1}^{K} \frac{1}{2} \frac{b_{\{x^*\}}^{\mathcal{A}\prime}{}_k^T b_{\{x^*\}}^{\mathcal{A}\prime}{}_k}{M+\lambda}.$$

We can similarly upper bound the gain of an arbitrary element:

$$\sum_{k=1}^{K} F_k(\mathcal{A} \cup \{x\}) - F_k(\mathcal{A}) \leq \sum_{k=1}^{K} \frac{1}{2} b_x^{\mathcal{A}\prime}{}_k^T (m C_x^{\mathcal{A}\prime} + \lambda I)^{-1} b_x^{\mathcal{A}\prime}{}_k,$$

where

$$C_{\mathcal{S}}^{\mathcal{A}\prime} = C_{\mathcal{S}} - C_{\mathcal{S}\mathcal{A}} \left(C_{\mathcal{A}} + \frac{\lambda}{m} I \right)^{-1} C_{\mathcal{A}\mathcal{S}}.$$

Again using the eigenvalue bound from Lemmas 5.2.1-5.2.3, we know that

$$b_{\mathcal{S}}^{\mathcal{A}\prime}{}_k^T (m C_{\mathcal{S}}^{\mathcal{A}\prime} + \lambda I)^{-1} b_{\mathcal{S}}^{\mathcal{A}\prime}{}_k \leq \frac{b_{\mathcal{S}}^{\mathcal{A}\prime}{}_k^T b_{\mathcal{S}}^{\mathcal{A}\prime}{}_k}{m\lambda_{\min}(C) + \lambda}.$$

Algorithm 5.4 Grouped Forward Regression

Given: elements $\mathcal{X}$, groups Γ, target $\mathcal{Y}$
Define F as in Equation (5.3)
Let $\mathcal{D}_0 = \emptyset$.
for $j = 1, \ldots$ **do**
  Let $\mathcal{G}^* = \arg\max_{\mathcal{G} \in \Gamma} \frac{F(\mathcal{D}_{j-1} \cup \mathcal{G}) - F(\mathcal{D}_{j-1})}{c(\mathcal{G})}$.
  Let $\mathcal{D}_j = \mathcal{D}_{j-1} \cup \mathcal{G}^*$.
end for

So now, setting $\alpha = \frac{m\lambda_{\min}(C)+\lambda}{M+\lambda}$ and $\beta = 0$ we have

$$\begin{aligned}
&\frac{F(\mathcal{D}_{j-1} \cup \{x^*\}) - F(\mathcal{D}_{|-\infty})}{c^*} \\
&= \sum_{k=1}^{K} \frac{F_k(\mathcal{D}_{j-1} \cup \{x^*\}) - F_k(\mathcal{D}_{|-\infty})}{c^*} \\
&\geq \sum_{k=1}^{K} \frac{1}{2} \frac{1}{M+\lambda} \frac{{b^{\mathcal{A}}_{\{x^*\}_k}}' {}^T b^{\mathcal{A}}_{\{x^*\}_k}{}'}{c(x^*)} \\
&\geq \max_x \sum_{k=1}^{K} \frac{1}{2} \frac{1}{M+\lambda} \frac{{b^{\mathcal{A}'}_{x\,k}}^T b^{\mathcal{A}'}_{x\,k}}{c(x)} \\
&\geq \max_x \frac{m\lambda_{\min}(C)+\lambda}{M+\lambda} \sum_{k=1}^{K} \frac{F_k(\mathcal{A} \cup \{x\}) - F_k(\mathcal{A})}{c(x)} \\
&= \max_x \frac{m\lambda_{\min}(C)+\lambda}{M+\lambda} \frac{F(\mathcal{A} \cup \{x\}) - F(\mathcal{A})}{c(x)}
\end{aligned}$$

completing the proof. ■

These two results together show that, for a given set of sparse approximation problems with identical approximation guarantees, such as a set of problems all with the same regularization parameters and variables, the resulting simultaneous sparse approximation problem has identical approximation guarantees to each individual problem as well.

5.6 Grouped Features

One final extension of the standard sparse approximation problem that we will examine is the *grouped* version of the problem. In this variant, each feature belongs to some group $\mathcal{G}$, and the

Algorithm 5.5 Grouped Orthogonal Matching Pursuit

Given: elements $\mathcal{X}$, groups Γ, target $\mathcal{Y}$
Define F as in Equation (5.3)
Let $\mathcal{D}_0 = \emptyset$.
for $j = 1, \ldots$ **do**
  Let $w^* = \arg\min_w \mathbb{E}[(Y - w^T X_{\mathcal{D}_{j-1}})^2]$
  Let $b_x^{\mathcal{D}_{j-1}} = \mathbb{E}[(Y - w^{*T} X_{\mathcal{D}_{j-1}})^T x]$.
  Let $\mathcal{G}^* = \arg\max_{\mathcal{G} \in \Gamma} \frac{b_{\mathcal{G}}^{\mathcal{D}_{j-1}{}^T} (C_{\mathcal{G}} + \lambda I)^{-1} b_{\mathcal{G}}^{\mathcal{D}_{j-1}}}{c(\mathcal{G})}$.
  Let $\mathcal{D}_j = \mathcal{D}_{j-1} \cup \mathcal{G}^*$.
end for

budget and costs $c(\mathcal{G})$ are defined over the groups selected, not the individual features. Selecting any one feature within the group is effectively equivalent to selecting the entire group. This scenario typically arises when features are computed using some common process or derived from some base feature.

Formally, we are given some set of groups $\Gamma = \{\mathcal{G}_1, \mathcal{G}_2, \ldots\}$ such that each group contains some set of the features $\mathcal{G} \subseteq \mathcal{X}$. We additionally assume that the sets form a partition of the set $\mathcal{X}$, so that $\mathcal{G} \cap \mathcal{G}' = \emptyset$ for all $\mathcal{G}, \mathcal{G}' \in \Gamma$ and $\mathcal{G} \neq \mathcal{G}'$.

Let F' be the grouped set function maximization, or equivalently, the corresponding set function over the raw variables in the selected groups:

$$F'(\Sigma) = F(\mathcal{S}(\Sigma)) \tag{5.18}$$

$$\mathcal{S}(\Sigma) = \bigcup_{\mathcal{G} \in \Sigma} \mathcal{G}. \tag{5.19}$$

One typical solution to solving this problem in practice is to use the standard Forward Regression and Orthogonal Matching Pursuit algorithms adapted to the group setting. In this approach the same greedy criteria over single features is used, but the entire group corresponding to the selected feature is used as the selected group. Effectively this approach greedily selects groups by using the max, or L_∞ norm of some criteria over the features in the group. Another obvious variant of the OMP algorithm is to use the L_2 norm of the OMP criteria evaluated over the features in the group.

To be concrete, the standard OMP criteria maximizes the gradient term:

$$b_x^{\mathcal{D}_{j-1}} = \mathbb{E}[(Y - w^{*T} X_{\mathcal{D}_{j-1}})^T x],$$

specifically the squared term:

$$b_x^{\mathcal{D}_{j-1}{}^T} b_x^{\mathcal{D}_{j-1}}.$$

The single feature version of the grouped OMP approach selects the group $\mathcal{G}$ which maximizes:

$$\left\| b_{\mathcal{G}}^{\mathcal{D}_{j-1}} \right\|_\infty^2,$$

while the other OMP variant maximizes

$$\left\| b_{\mathcal{G}}^{\mathcal{D}_{j-1}} \right\|_2^2.$$

There are simple counter-examples that show where both of these approaches fail. For the L_∞ approach, the algorithm will pick groups that have one single good feature, while other groups may contain arbitrarily many good, but slightly worse, features. The L_2 approach fails in an opposite fashion. Given a group with many identical copies of the same feature, the L_2 criteria will be very high, when the group may in fact have very little benefit.

We propose two group-based variants of the FR and OMP algorithms, given in Algorithm 5.4 and Algorithm 5.5 to fix these problems. The FR variant of the algorithm is a very natural greedy strategy evaluated over the groups instead of the individual features. The OMP approach uses a method similar to the L_2 criteria proposed above, but modifies this by instead computing the quadratic form

$$b_{\mathcal{G}}^{\mathcal{D}_{j-1}{}^T} \left(C_{\mathcal{G}} + \lambda I \right)^{-1} b_{\mathcal{G}}^{\mathcal{D}_{j-1}}.$$

An alternative way to view this OMP variant is that it is identical to the L_2 approach, but with the added caveat that the data is whitened within each group prior to running the algorithm.

We do not present approximation guarantees here, but we propose that similar approximation guarantees to the OMP and FR algorithms in the standard sparse approximation setting can likely be extended to this setting as well.

5.7 Experimental Results

We now present a number of results on practical applications. In out results we will compare the optimal feature set for a given budget (OPT), Forward Regression (FR), Orthogonal Matching Pursuit (OMP), and Lasso (L1). For FR and OMP, we use the cost-greedy variants presented here, as well as comparisons against the original implementations for uniform feature costs [Pati et al., 1993, Miller, 2002].

For the Lasso approach, we utilize two different approaches. For strictly regression settings we use Least Angle Regression [Efron et al., 2004] as implemented by the `lars` R package [Hastie and Efron, 2013]. For the multiclass experiments that will follow, we utlize the L1 regularization feature of Vowpal Wabbit [Langford, 2013] to generate the Lasso regularization path. Finally, in the Lasso approaches, we take the set of variables given by the L1 regularization path, and retrain the model without the L1 constraint to obtain the optimal performance for a given budget.

For datasets, we use the UCI Machine Learning Repository [Frank and Asuncion, 2010], specifically the 'housing' and 'wine quality' datasets for regression problems and the 'pendigits' and 'letter' datasets for multiclass classification problems.

For all problems we normalize the features to unit variance and zero mean, and for regression problems we normalize the target as well. The 'housing' dataset uses $d = 13$ features, while the

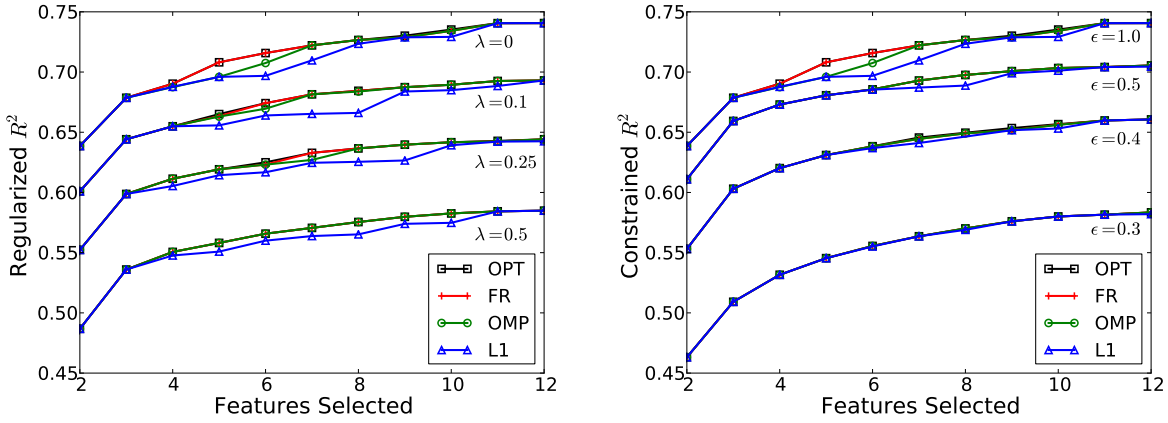


Figure 5.2: Performance of optimal (OPT), Forward Regression (FR), Orthogonal Matching Pursuit (OMP) and Lasso (L1) algorithms on the UCI 'housing' dataset, for various parameterizations of the regularized (left) and constrained (right) sparse approximation problems.

'wine quality' dataset has $d = 11$ features. For the multiclass problems, 'pendigits' has $d = 16$ features and $k = 10$ classes, while 'letter' has $d = 16$ features and $k = 26$ classes.

Figure 5.2 shows the results for all algorithms on the 'housing' dataset using uniform costs, while varying the regularization and constraint parameters λ and ϵ for the regularized and constrained sparse approximation problems. We use $\lambda \in \{0, 0.1, 0.25, 0.5\}$ and $\epsilon \in \{1, 0.5, 0.4, 0.3\}$ here. As predicted by our bounds, we see the greedy algorithms converge to optimal performance as the constraint or regularization increasingly penalizes larger weights. We also observe that the Lasso algorithm converges as the weight vector is increasingly penalized, but is consistently outperformed by the greedy approaches.

Figure 5.3 demonstrates the performance of all algorithms for the two regression datasets on the budgeted sparse approximation problem. Here we use synthetic costs, as no costs are provided with the datasets used. The costs are sampled from a gamma distribution with parameters $k = 2, \theta = 2.0$, to ensure that there are a mix of expensive and cheap costs. All results are average over 20 different sets of sampled costs.

To obtain a cost-based or budgeted version of the Lasso algorithm, we scale each feature x by the inverse of the cost $c(x)$, effectively scaling the weight which is penalized in the L_1 term by $c(x)$, giving a cost-scaled version of the L_1 norm. This approach requires that you ensure that the Lasso implementation used is not re-normalizing the features at any point, which we have done.

We see the same behavior here that we saw in the first comparison and in our theoretical bounds. Namely, that Forward Regression is the closest to optimal performance, followed by Orthogonal Matching Pursuit, with the Lasso approach typically giving the worst performance.

For the smooth and simultaneous sparse approximation settings, Figure 5.4 demonstrates cost-greedy algorithms as well as their uniform cost counterparts on two multiclass datasets. Here we use the one-vs-all approach to multiclass classification, using the logistic loss for each of the k

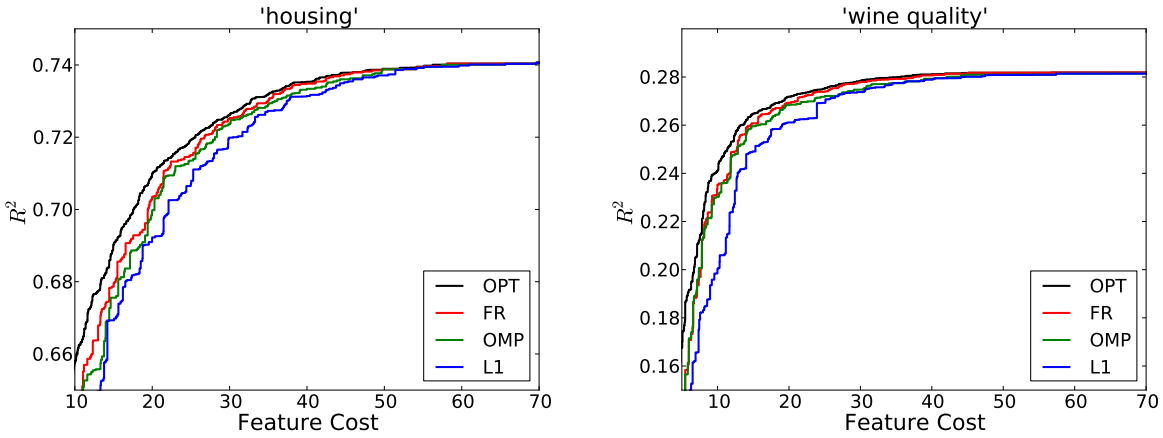


Figure 5.3: Performance of optimal (OPT), Forward Regression (FR), Orthogonal Matching Pursuit (OMP) and Lasso (L1) algorithms on the UCI 'housing' and 'wine quality' datasets, for the budgeted sparse approximation problem using synthetic costs. Results are averaged over 20 sets of sampled synthetic costs.

resulting binary classification problems. This gives an overall sparse approximation problem with k simultaneous smooth losses with strong smoothness parameter $M = 1$.

For feature costs, we use the same synthetic cost sampling procedure described above, also averaged over 20 different sets of costs. In this problem we see the same behavior observed in Figure 5.3 with Forward Regression giving the best overall trade-off of cost and accuracy, followed by Orthogonal Matching Pursuit and then Lasso. We do not compare to optimal performance for these experiments because it is significantly more expensive to train a logistic regressor than a regular least squares fit for all subsets of the the features.

Additionally, these results show that using the cost-greedy variant of each algorithm is critical to obtaining good performance on the budgeted problem. The variants intended for use in the uniform cost case significantly underperform in this setting. Although we do not know of any analysis for the budgeted setting, our results also indicate that a version of the Lasso algorithm which uses a weighted L_1 norm significantly outperforms the traditional unweighted approach when dealing with budgeted feature selection problems.

As another example of a budgeted feature selection problem, we use the Yahoo! Learning to Rank Challenge dataset augmented with feature costs [Xu et al., 2012]. Though this is a ranking problem, we use the regression version of the problem. Each document in the dataset is paired with a relevance ranking in $\{0, 1, 2, 3, 4\}$, and we use the normalized vector of relevances as the regression target Y. The dataset consists of 519 features, with costs drawn from $\{1, 5, 10, 20, 50, 100, 150, 200\}$. The full training dataset contains 473134 examples, but we use only the first 200000 as the Lasso implementation used in these experiments required the full dataset to be stored in memory, and our test machine did not have enough memory for the complete dataset.

Figure 5.5 gives the results of both cost-greedy and uniform variants of all algorithms on this

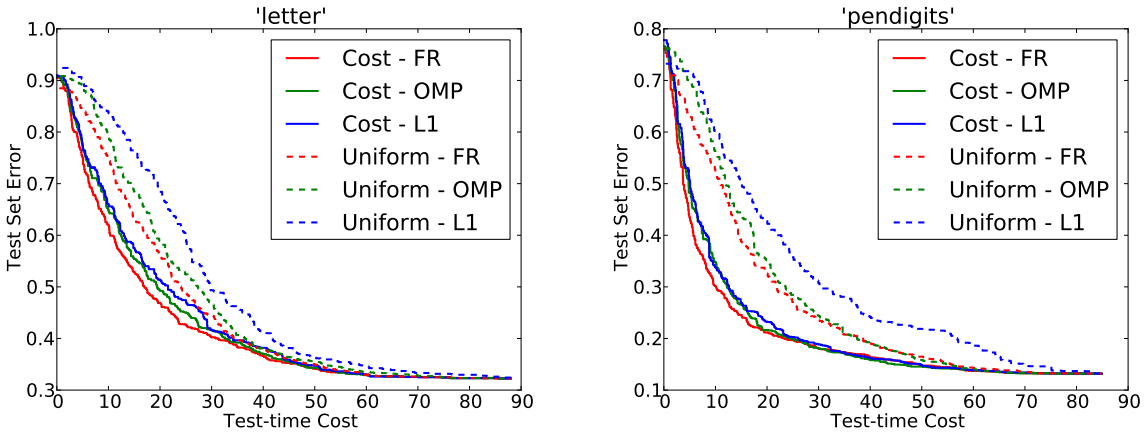


Figure 5.4: Performance of cost-greedy and uniform cost variants of Forward Regression (FR), Orthogonal Matching Pursuit (OMP) and Lasso (L1) algorithms on the UCI 'letter' and 'pendigits' datasets, for the budgeted sparse approximation problem using synthetic costs. Results are averaged over 20 sets of sampled synthetic costs.

dataset. Here we see the same basic behavior as all previous datasets, with a slightly less pronounced advantage to the cost-greedy variants over the uniform cost algorithms. Figure 5.6 gives a comparison of the training time required for all algorithms, as a function of the number of features selected. Note that in the regression case, all algorithms first compute the covariance matrix C and vector b, and then operate on this reduced ($d = 519$ dimensions) space, so the training time should not scale with the number of training examples, except for the initial fixed cost (here 14.41s) of computing these quantities. Overall we observe that the OMP and L1 algorithms are fairly efficient, even for large numbers of features, while the FR algorithm is nearly two orders of magnitude more expensive.

Finally, we present experimental results for the grouped feature setting. In this setting we contrast the proposed grouped FR (Algorithm 5.4) and grouped OMP (Algorithm 5.5) algorithms, the optimal group selections, the single feature versions of the FR and OMP algorithms (the L_∞ approach described in Section 5.6) and the L_2 version of OMP, which is equivalent to the proposed algorithm with the added assumption that the data is already whitened within groups.

For the first dataset we use the same Yahoo Learning to Rank data from the previous results, but with randomly generated groups (Figure 5.7). We randomly distribute the $d = 519$ features evenly among 17 groups of approximately 30 features each. All results are averaged over 20 randomly sampled sets of groups. The second dataset used is synthetically sampled data (Figure 5.8). We generate $d = 160$ features for 100 examples randomly, such that the features have moderately high correlations of 0.6, in a manner similar to Das and Kempe [2011]. The groups are also randomly sampled with 16 groups of 10 features each. We then randomly sample half of the groups (8) and use uniform weights over the features in those groups to construct the target vector, along with added noise ($\sigma = 0.1$).

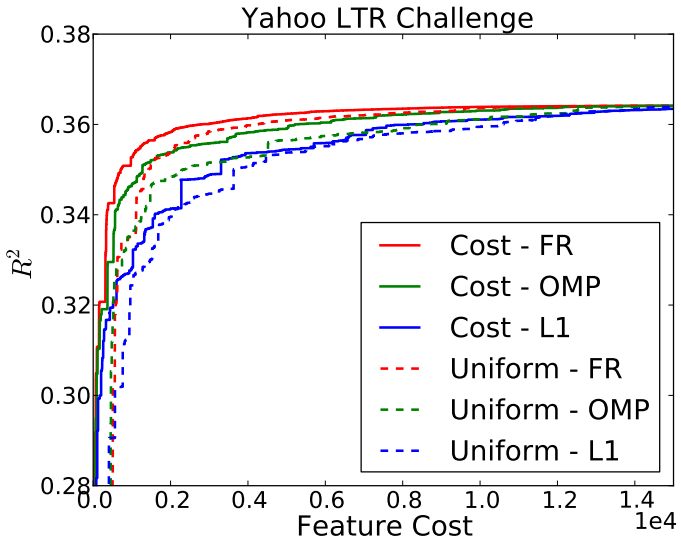


Figure 5.5: Performance of cost-greedy and uniform cost variants of Forward Regression (FR), Orthogonal Matching Pursuit (OMP) and Lasso (L1) algorithms on the budgeted version of the Yahoo! Learning to Rank dataset.

In these experiments we see that the grouped version of the FR algorithm clearly dominates and is closest to optimal, while the grouped version of the OMP algorithm is slightly worse. The L_2 variant, or non-whitened version of the grouped OMP approach is substantially worse than all algorithms. The L_∞ or single feature versions are significantly sub-optimal for small numbers of selected groups, due to their inability to measure the overall benefit of the groups, but can sometimes (synthetic data) outperform the grouped approaches for larger numbers of selected groups.

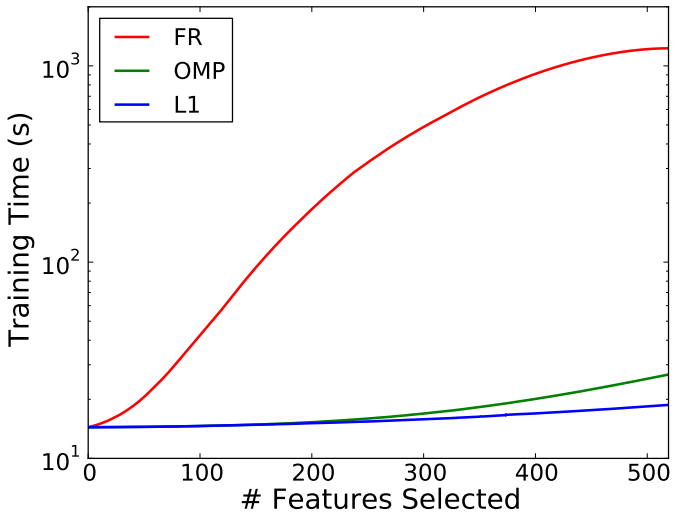


Figure 5.6: A comparison of the training time required for various numbers of selected features for Forward Regression (FR), Orthogonal Matching Pursuit (OMP) and Lasso (L1) algorithms on the Yahoo! Learning to Rank dataset. Note the log-scale of training time.

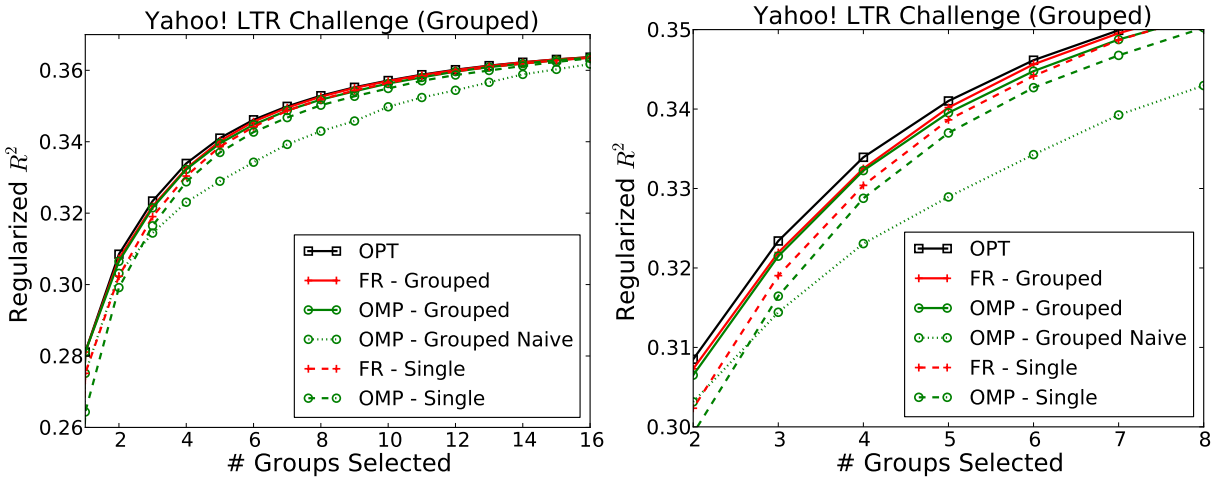


Figure 5.7: Comparison of group selection approaches for the grouped feature selection problem with randomly sampled groups for the Yahoo! Learning to Rank data (left) along with a zoomed portion of the results (right). Algorithms compared are the grouped FR and OMP variants, single feature or L_∞ versions, and the un-whitened or L_2 variant of the OMP approach.

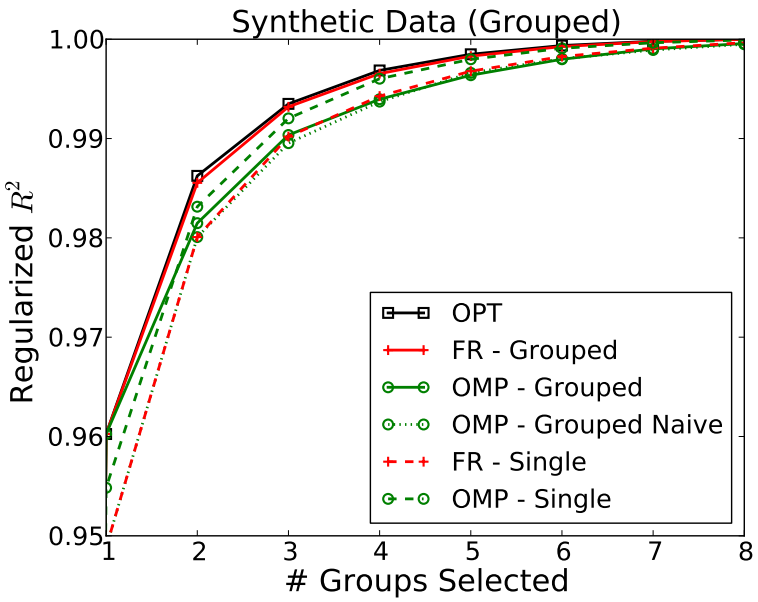


Figure 5.8: Comparison of group selection approaches for the grouped feature selection problem with synthetic data. Algorithms compared are the grouped FR and OMP variants, single feature or L_∞ versions, and the un-whitened or L_2 variant of the OMP approach.

Part III

Anytime Prediction

Chapter 6

SPEEDBOOST: Anytime Prediction Algorithms

In this chapter we will now combine the algorithms and analysis from the previous chapters to tackle the anytime prediction problem we originally set out to study. Specifically, we will combine the widely applicable framework of functional gradient methods with the cost-greedy algorithms and theoretical guarantees from our analysis of sparse approximation methods, to obtain methods for building ensemble predictors with similar near-optimal guarantees.

6.1 Background

Recall the desirable properties for anytime algorithms given by Zilberstein [1996]:

- Interruptability: a prediction can be generated at any time.
- Monotonicity: the quality of a prediction is non-decreasing over time.
- Diminishing Returns: prediction quality improves fastest at early stages.

To accomplish these goals we will rely heavily on the two major areas examined in the earlier parts of this work. To obtain the incremental, interruptable behavior we would like for updating predictions over time we will learn an additive ensemble of weaker predictors. This aspect of our anytime approach is based on the functional gradient framework detailed in Chapter 2. By learning a sequence of weak predictors, represented as a linear combination of functions h, we naturally have an interruptable predictor. We simply evaluate the weak predictors in sequence and compute the linear combination of the outputs whenever a prediction is desired, allowing for predictions to be updated over time.

Building on that foundation, we will then introduce the cost-greedy strategies studied in Chapters 4 and 5. This augmented cost-greedy version of functional gradient methods is simply an

extension of the sparse approximation setting discussed in the previous chapter, with each weak predictor representing a variable to be selected. As we have shown previously, using a cost-greedy approach ensures that we select sequences of weak predictors that behave near-optimally and increase accuracy as efficiently as possible, satisfying the last two properties.

We will now develop similar algorithms to the Forward Regression (FR) and Orthogonal Matching Pursuit (OMP) algorithms discussed in Chapter 5 specifically for the functional gradient domain, which will generate sequences of weak predictors h that obtain good anytime performance across a range of computational budgets. Specifically, we will show in Section 6.4 that the theoretical results for sparse approximation studied in Chapter 5 all generalize to certain variants of our anytime approach.

6.2 Anytime Prediction Framework

Building from the functional gradient framework discussed in Chapter 2, we will base our framework for anytime prediction around the additive, incremental predictors discussed there. In the anytime setting, as in the functional gradient setting, we consider predictors $f : \mathcal{X} \to \mathcal{V}$ which compute some prediction $f(x) \in \mathcal{V}$ for inputs $x \in \mathcal{X}$

To obtain the incremental improvement in performance we desire for our anytime learner, we use the additive predictors f from functional gradient methods, which are a weighted combination of weaker predictors $h \in \mathcal{H}$

$$f(x) = \sum_i \alpha_i h_i(x), \tag{6.1}$$

where $\alpha_i \in \mathbb{R}$ and $h_i : \mathcal{X} \to \mathcal{V}$.

We will use the same loss function optimization setting as functional gradient methods as well. Recall that we wish to minimize some objective functional $\mathcal{R}$:

$$\min_f \mathcal{R}[f].$$

Typically, $\mathcal{R}$ is some pointwise loss, evaluated over training data

$$\mathcal{R}[f] = \sum_{n=1}^{N} \ell(f(x_n)), \tag{6.2}$$

but as discussed in Chapter 2, a number of other objective functionals are possible.

We will model the time budget portion of the anytime setting as a budget constraint, in the same manner as the budgeted maximization problems discussed in Chapter 4 and Chapter 5. We assume that each weak predictor h has an associated measure of complexity, or cost, $\tau(h)$ where $\tau : \mathcal{H} \to \mathbb{R}$. This measure of complexity allows for weak predictors which trade accuracy for computational efficiency and vice versa.

For the case where each predictor h can have variable computational cost per example, such as a decision tree, we use the expected computation time. Let $\tau_x(h)$ be the cost of evaluating h on example x. Then:

$$\tau(h) = \mathbb{E}_{\mathcal{X}}[\tau_x(h)].$$

We further assume that calculating the sum of the predictor outputs weighted by α_t takes negligible computation, so that the total computation time is dominated by the computation time of each predictor. Using this complexity measure we can describe the predictions generated at a given time $\mathcal{T}$ as

$$f_{\langle \mathcal{T} \rangle} = \sum_{i=1}^{i^*} \alpha_i h_i(x), \quad i^* = \max \left\{ i' \;\middle|\; \sum_{i=1}^{i'} \tau(h_i) < \mathcal{T} \right\}.$$

In the anytime setting, we will then attempt to optimize the performance $\mathcal{R}[f_{\langle \mathcal{T} \rangle}]$ for all budgets $\mathcal{T}$.

6.3 SPEEDBOOST

We now consider learning algorithms for generating anytime predictors. Formally, given a set of weak predictors $\mathcal{H}$ we want to find a sequence of weights and predictors $\{\alpha_i, h_i\}_{i=1}^{\infty}$ such that the predictor f constructed in Equation (6.1) achieves good performance $\mathcal{R}[f_{\langle \mathcal{T} \rangle}]$ at all possible stopping times $\mathcal{T}$.

Recall the properties of interruptability, monotonicity, and diminishing returns that are desirable in the anytime setting. An ensemble predictor as formulated in Section 6.2 naturally satisfies the interruptability property, by evaluating the weak predictors in sequence and stopping when necessary to output the final prediction.

Algorithm 6.1 SPEEDBOOST

Given: starting point f_0, objective $\mathcal{R}$
for $i = 1, \ldots$ **do**
  Let $h_i, \alpha_i = \arg\max_{h \in \mathcal{H}, \alpha \in \mathbb{R}} \frac{[\mathcal{R}[f_{i-1}] - \mathcal{R}[f_{i-1} + \alpha h]]}{\tau(h)}$.
  Let $f_i = f_{i-1} + \alpha_i h_i$.
end for
return Predictor $(\{(h_i, \alpha_i)\}_i)$

To learn predictors which satisfy the last two properties, we present SPEEDBOOST (Algorithm 6.1), a natural greedy selection approach for selecting weak predictors. This algorithm uses a cost-greedy selection procedure to select the weak learner h which gives the largest gain in objective $\mathcal{R}$ per unit-cost:

$$\arg\max_{h \in \mathcal{H}, \alpha \in \mathbb{R}} \frac{[\mathcal{R}[f_{i-1}] - \mathcal{R}[f_{i-1} + \alpha h]]}{\tau(h)}. \tag{6.3}$$

It can be shown that the non-cost based version of this optimization,

$$\arg\max_{h\in\mathcal{H},\alpha\in\mathbb{R}} \left[\mathcal{R}[f_{i-1}] - \mathcal{R}[f_{i-1} + \alpha h]\right], \tag{6.4}$$

is the same optimization performed by many boosting algorithms implicitly. In many functional gradient methods, we select the predictor which maximizes

$$\arg\max_{h\in\mathcal{H}} \frac{\langle \nabla, h\rangle}{\|h\|},$$

as per the discussion in Chapter 2. This optimization is equivalent to minimizing Equation (6.4) for many losses. For example, in squared error regression, or in the exponential loss optimization of AdaBoost [Freund and Schapire, 1997], the two are equivalent.

As we will discuss later in Section 6.4, this algorithm is very similar to the Forward Regression algorithm discussed in the previous chapter (Algorithm 5.1), with one exception. In this algorithm, the weight is optimized only over the newly added element h_i, while in Forward Regression, the weights are re-optimized over all selected variables.

SPEEDBOOST will select a sequence of feature functions h that greedily maximize the improvement in the algorithm's prediction per unit time. By using a large set $\mathcal{H}$ of different types of weak predictors with varying time complexity, this algorithm provides a simple way to trade computation time with improvement in prediction accuracy. Unfortunately, for many classes of functions where $\mathcal{H}$ is very large, Algorithm 6.1 can be impractical. Furthermore, unlike in regular boosting, where the projection operation can be implemented as an efficient learning algorithm, the cost-greedy selection criteria in Equation (6.3) is not so easily optimizable.

To address this issue, we use the weak learner selection methods of functional gradient descent and other boosting methods. As shown in Chapter 2, we can implement an efficient gradient projection operation to select good candidate weak predictors.

Recall from Section 2.2 that, given a function ∇ representing the functional gradient (Section 2.1.2), the projection of ∇ on to a set of weak predictors $\mathcal{H}$ is defined using the functional inner product,

$$\begin{aligned} \text{Proj}\,(\nabla, \mathcal{H}) &= \arg\max_{h\in\mathcal{H}} \frac{\langle \nabla, h\rangle}{\|h\|} \\ &= \arg\max_{h\in\mathcal{H}} \frac{\sum_{n=1}^{N} \nabla(x_n) h(x_n)}{\sum_{n=1}^{N} h(x_n)^2}. \end{aligned} \tag{6.5}$$

For classifiers with outputs in $h(x) \in \{-1, +1\}$, Equation (6.5) is simply a weighted classification problem. Equivalently, when $\mathcal{H}$ is closed under scalar multiplication, the projection rule can minimize the norm in function space,

$$\begin{aligned} \text{Proj}\,(\nabla, \mathcal{H}) &= \arg\min_{h\in\mathcal{H}} \|\nabla - h\|^2 \\ &= \arg\min_{h\in\mathcal{H}} \sum_{n=1}^{N} (h(x_n) - \nabla(x_n))^2, \end{aligned} \tag{6.6}$$

which corresponds directly to solving the least squares regression problem.

Algorithm 6.2 SPEEDBOOST.Proj

Given: starting point f_0, objective $\mathcal{R}$
for $i = 1, \ldots$ **do**
  Compute gradient $\nabla_i = \nabla\mathcal{R}[f_{i-1}]$.
  Let $\mathcal{H}^* = \left\{h_j^* \;\middle|\; h_j^* = \text{Proj}\left(\nabla_i, \mathcal{H}_j\right)\right\}$.
  Let $h_i, \alpha_i = \arg\max_{h \in \mathcal{H}^*, \alpha \in \mathbb{R}} \frac{[\mathcal{R}[f_{i-1}] - \mathcal{R}[f_{i-1} + \alpha h]]}{\tau(h)}$
  Let $f_i = f_{i-1} + \alpha_i h_i$.
end for
return Predictor $\left(\{(h_i, \alpha_i)\}_i\right)$

Algorithm 6.2 gives a more tractable version of SPEEDBOOST for learning anytime predictors based on the projection strategy of functional gradient descent. Here we assume that there exist a relatively small number of weak learning algorithms, $\{\mathcal{H}_1, \mathcal{H}_2, \ldots\}$ representing classes of functions with similar complexity. For example, the classes may represent decision trees of varying depths or kernel-based learners of varying complexity. The algorithm first projects the functional gradient onto each individual class as in gradient boosting, and then uses the best result from each class to perform the greedy selection process described previously.

Algorithm 6.3 SPEEDBOOST.MP

Given: starting point f_0, objective $\mathcal{R}$
for $i = 1, \ldots$ **do**
  Compute gradient $\nabla_i = \nabla\mathcal{R}[f_{i-1}]$.
  Let $\mathcal{H}^* = \left\{h_j^* \;\middle|\; h_j^* = \text{Proj}\left(\nabla_i, \mathcal{H}_j\right)\right\}$.
  Let $h_i = \arg\max_{h \in \mathcal{H}^*} \frac{\langle \nabla_i, h \rangle^2}{\tau(h)}$.
  Let $\alpha_i = \arg\min_{\alpha \in \mathbb{R}} \mathcal{R}[f_{i-1} + \alpha h_i]$
  Let $f_i = f_{i-1} + \alpha_i h_i$.
end for
return Predictor $\left(\{(h_i, \alpha_i)\}_i\right)$

This modification to the greedy algorithm is closely related to another greedy feature selection algorithm. Algorithm 6.3 gives a complexity-weighted version of Matching Pursuit [Mallat and Zhang, 1993], adapted to function spaces. In this algorithm, we use the selection criteria

$$\arg\max_{h \in \mathcal{H}^*} \frac{\langle \nabla_i, h \rangle^2}{\tau(h)},$$

to select the next weak predictor at each iteration. This selection criteria is equivalent to the criteria used in Orthogonal Matching Pursuit [Pati et al., 1993], discussed in Chapter 5, but the algorithm

only fits the weight α_i, instead of refitting the weights on all of the weak learners. In the next section we will discuss the OMP equivalent for this algorithm, and the theoretical guarantees that can be derived for that algorithm (Algorithm 6.5).

In practice we use Algorithm 6.2 in favor of Algorithm 6.3 because the linesearch and loss evaluation required for that selection criteria is typically not significantly more expensive than the Matching Pursuit selection criteria. In the SPEEDBOOST.MP variant, we will require a linesearch anyway after the next weak predictor is selected, so computing it for each complexity class is not a problem. Furthermore, the projection of the gradient on to each complexity class $\mathcal{H}_j$ is typically much more expensive than the linesearch required to optimize the selection criteria in SPEEDBOOST.Proj, so that portion of the optimization does not contribute significantly to the training time.

6.4 Theoretical Guarantees

Algorithm 6.4 SPEEDBOOST.FR

Given: starting point f_0, objective $\mathcal{R}$
for $i = 1, \ldots$ **do**
  Let $h_i, \boldsymbol{\alpha}_i = \underset{h \in \mathcal{H}, \boldsymbol{\alpha} \in \mathbb{R}^i}{\arg\max} \frac{\left[\mathcal{R}[f_{i-1}] - \mathcal{R}[\sum_{j=1}^{i-1} \alpha_j h_j + \alpha_i h]\right]}{\tau(h)}$.
  Let $f_i = \sum_{j=1}^{i} \alpha_{ij} h_j$.
end for
return Predictor $(\{(h_i, \boldsymbol{\alpha}_i)\}_{i=1})$

Algorithm 6.5 SPEEDBOOST.OMP

Given: starting point f_0, objective $\mathcal{R}$
for $i = 1, \ldots$ **do**
  Compute gradient $\nabla_i = \nabla \mathcal{R}[f_{i-1}]$.
  Let $\mathcal{H}^* = \left\{ h_j^* \;\middle|\; h_j^* = \text{Proj}\left(\nabla_i, \mathcal{H}_j\right) \right\}$.
  Let $h_i = \underset{h \in \mathcal{H}^*}{\arg\max} \frac{\langle \nabla_i, h \rangle^2}{\tau(h)}$.
  Let $\boldsymbol{\alpha}_i = \underset{\boldsymbol{\alpha} \in \mathbb{R}^i}{\arg\min}\, \mathcal{R}[\sum_{j=1}^{i-1} \alpha_j h_j + \alpha_i h_i]$.
  Let $f_i = \sum_{j=1}^{i} \alpha_{ij} h_j$.
end for
return Predictor $(\{(h_i, \boldsymbol{\alpha}_i)\}_{i=1})$

We will now analyze a variant of the SPEEDBOOST algorithm and prove that the predictor produced by this algorithm is near optimal with respect to any sequence of weak predictors that

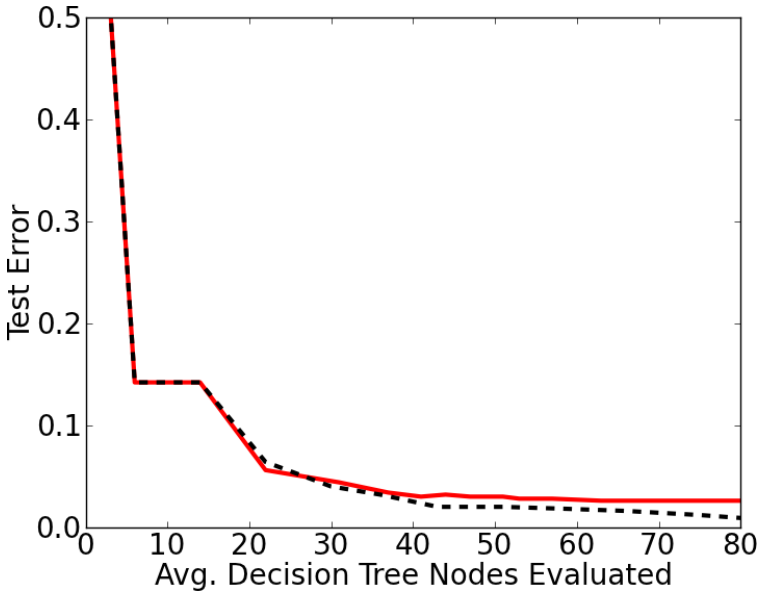


Figure 6.1: Test set error as a function of complexity for the UCI 'pendigits' dataset, comparing SPEEDBOOST.MP (Algorithm 6.3) (black dashed line) to SPEEDBOOST.OMP (Algorithm 6.5) (solid red line).

could be computed in the same amount of time, for a common set of loss functions and certain classes of weak predictors.

For analysis, one can interpret Algorithm 6.3, and to some degree Algorithm 6.2, as a time-based version of Matching Pursuit [Mallat and Zhang, 1993]. Unfortunately, the sequence of weak predictors selected by matching pursuit can perform poorly with respect to the optimal sequence for some fixed time budget $\mathcal{T}$ when faced with highly correlated weak predictors [Pati et al., 1993]. A modification of the Matching Pursuit algorithm called Orthogonal Matching Pursuit [Pati et al., 1993] addresses this flaw.

As discussed previously, the main different between the MP and OMP approach is the behavior of the weight fitting at each iteration. In SPEEDBOOST.MP we fit the only the weight α_i on weak predictor h_i at each iteration. In the OMP approach and in SPEEDBOOST.OMP, we refit all the weights α on each weak predictor at each iteration. Similarly, the basic SPEEDBOOST algorithm (Algorithm 6.1) differs only from Forward Regression in this same weight refitting aspect.

To that end, we present SPEEDBOOST.FR (Algorithm 6.4) and SPEEDBOOST.OMP (Algorithm 6.5) which are modifications of the SPEEDBOOST (Algorithm 6.1) and SPEEDBOOST.MP (Algorithm 6.3), respectively. The key difference between these algorithms is the refitting of the weights on every weak predictor selected so far at every iteration of the algorithm.

The key disadvantage of using this algorithm in practice is that the output of all previous weak predictors must be maintained and the linear combination re-computed whenever a final prediction is desired.

In practice, we found that SPEEDBOOST and SPEEDBOOST.MP performed nearly as well as

Algorithm 6.5 in terms of improvement in the objective function, while being significantly cheaper to implement. We did not test SPEEDBOOST.FR, because it is intractable in practice for the boosting setting. To properly implement it would require enumerating all possible weak predictors in a given set and running a full weight optimization over all previous weak learners for each one, which would be prohibitively expensive.

Figure 6.1 shows a comparison of the test error on the UCI 'covertype' dataset for SPEEDBOOST.MP and SPEEDBOOST.OMP. In this case, while the training objective performances were nearly indistinguishable (not shown), Algorithm 6.5 overfit to the training data much more rapidly.

6.4.1 Uniformly Anytime Near-Optimality

Now that we have these function space equivalents of the FR and OMP algorithms, we can use the results in Chapter 5 to obtain approximation guarantees on them.

Assume that $\mathcal{R}$ is the pointwise loss given in Equation (6.2). Let $\mathcal{S}$ be a set of selected weak predictors $\mathcal{S} \subset \mathcal{H}$, and let $f_{\mathcal{S}}$ be the linear combination of those selected weak predictors which minimizes $\mathcal{R}$:

$$\begin{aligned} f_{\mathcal{S}} &= \sum_{i \in \mathcal{S}} \alpha_i^* h_i \\ \alpha^* &= \arg\min_{\boldsymbol{\alpha}} \mathcal{R}[\sum_{i \in \mathcal{S}} \alpha_i h_i]. \end{aligned} \tag{6.7}$$

Then, we can define a set function equivalent of minimizing the objective $\mathcal{R}$ as

$$\begin{aligned} F_{\mathcal{R}}(\mathcal{S}) &= \mathbb{E}_{\mathcal{X}}[\ell(0)] - \min_{\boldsymbol{\alpha} \in \mathbb{R}^{|\mathcal{S}|}} \mathbb{E}_{\mathcal{X}}[\ell(\sum_{i \in \mathcal{S}} \alpha_i h_i(x))] \\ &= \mathcal{R}[0] - \mathcal{R}[f_{\mathcal{S}}]. \end{aligned}$$

We have now reduced the anytime framework and SPEEDBOOST approach described above to the sparse approximation problem analyzed in Chapter 5. It can further be shown that SPEEDBOOST.FR (Algorithm 6.4) and SPEEDBOOST.OMP (Algorithm 6.5) are exactly equivalent to the Forward Regression and Orthogonal Matching Pursuit algorithms previously analyzed for the sparse approximation problem.

Using this reduction, we can apply all the results derived in Chapter 4 and Chapter 5 to our anytime prediction setting and boosting framework. For example, consider the regularized variant of the sparse approximation reduction given above:

$$F_{\mathcal{R}}(\mathcal{S}) = \mathbb{E}_{\mathcal{X}}[\ell(0)] - \min_{\boldsymbol{\alpha} \in \mathbb{R}^{|\mathcal{S}|}} \mathbb{E}_{\mathcal{X}}[\ell(\sum_{i \in \mathcal{S}} \alpha_i h_i(x)) + \frac{\lambda}{2} \boldsymbol{\alpha}^T \boldsymbol{\alpha}]. \tag{6.8}$$

This is equivalent to using a modified optimization problem where the weights on the weak

learners are regularized:

$$\min_f \mathcal{R}[f] + \frac{\lambda}{2}\boldsymbol{\alpha}^T\boldsymbol{\alpha}$$
$$f = \sum_i \alpha_i h_i.$$

We can now apply the bounds for the regularized, smooth sparse approximation problem to this setting and get a guarantee for the performance of our SPEEDBOOST variants.

Theorem 6.4.1 (Uniformly Anytime Approximation Guarantee)**.** *Let $F_{\mathcal{R}}$ be the regularized version of the anytime problem given in Equation (6.8), with regularization parameter λ. Assume that the weak predictors in $\mathcal{H}$ all have bounded norm $\|h\| \leq 1$. Let loss ℓ be an M-strongly smooth functional. Let S be any sequence of elements in $\mathcal{H}$. Let $\gamma = \frac{\lambda}{M+\lambda}$. Algorithm 6.4 selects a sequence of weak predictors $G = \{h_i \mid h_i \in \mathcal{H}\}_i$ such that for any time $\mathcal{T} = \sum_{i=1}^{i'} \tau(h_i)$,*

$$F_{\mathcal{R}}(\mathcal{G}_{\langle\mathcal{T}\rangle}) > \left(1 - e^{-\gamma}\right) F_{\mathcal{R}}(S_{\langle\mathcal{T}\rangle}),$$

and Algorithm 6.5 selects a sequence of weak predictors $G' = \{h'_i \mid h'_i \in \mathcal{H}\}_i$ such that for any time $\mathcal{T}' = \sum_{i=1}^{i'} \tau(h'_i)$,

$$F_{\mathcal{R}}(\mathcal{G}'_{\langle\mathcal{T}'\rangle}) > \left(1 - e^{-\gamma^2}\right) F_{\mathcal{R}}(S_{\langle\mathcal{T}'\rangle}).$$

■

The proof is a direct application of the bounds in the previous chapters to the sparse approximation reduction we've detailed above.

Theorem 6.4.1 states that, for all times $\mathcal{T}$ that correspond to the computation times that weak learners selected by SPEEDBOOST.FR and SPEEDBOOST.OMP update their prediction, the resulting improvement in loss $\mathcal{R}$ is approximately as large as any other sequence of weak learners that could have been computed up to that point. This means that the anytime predictor generated by those algorithms is competitive even with sequences specifically targeting fixed time budgets, uniformly across all times at which the anytime predictor computes new predictions.

Other results from previous chapters could also be easily extended to this setting using the above sparse approximation reduction. For example, the doubling algorithm discussed in Section 4.4 for obtaining a bi-criteria approximation with respect to any arbitrary budget $\mathcal{T}$ could be adapted to SPEEDBOOST and its variants to obtain the same bi-criteria approximation guarantees for arbitrary time budgets.

6.5 Experimental Results

6.5.1 Classification

Our first application is a set of classification problems from the UCI Machine Learning Repository [Frank and Asuncion, 2010]. We use the multiclass extension [Mukherjee and Schapire, 2010] to the exponential loss

$$\ell_n(f(x_n)) = \sum_{k \neq y_n} \exp(f(x_n)_k - f(x_n)_{y_n}).$$

For weak predictors we use decision trees of varying depth up to 20 nodes deep. We use Algorithm 6.2 and the weighted classification form of gradient projection to select the sequence of trees for our anytime prediction algorithm.

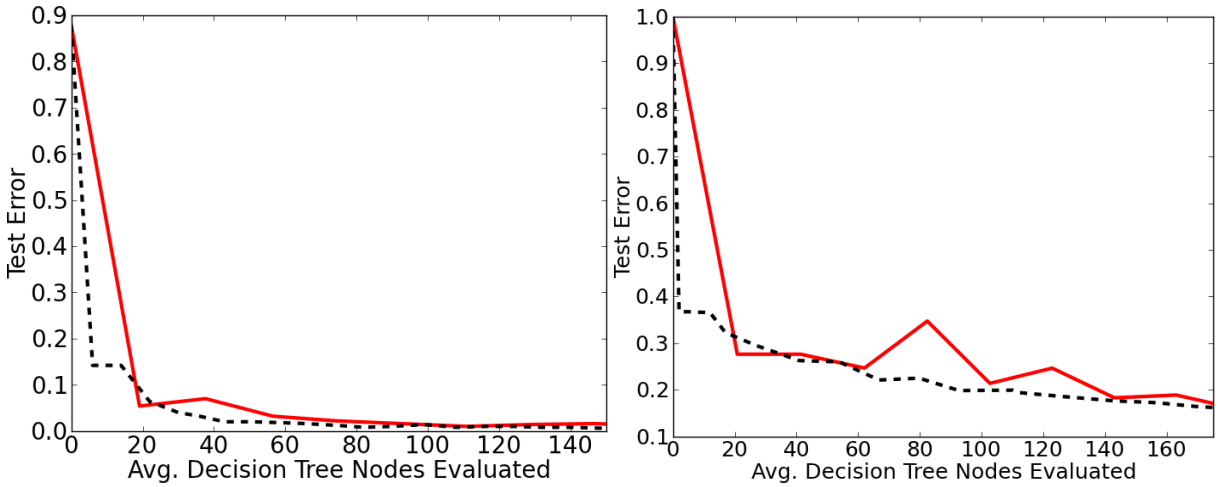


Figure 6.2: Test set error as a function of prediction time for the UCI 'pendigits' (top) and 'covertype' (bottom) dataset. The algorithms shown are SPEEDBOOST.Proj (black dashed line), and AdaBoost.MM [Mukherjee and Schapire, 2010] (red solid line).

As a point of comparison we use the AdaBoost.MM [Mukherjee and Schapire, 2010] implementation of multiclass boosting on the same set of trees. AdaBoost, when used in this manner to generate an anytime predictor, is effectively a variant on the greedy selection algorithm (Algorithm 6.1) which does not consider the computation time $\tau(h)$ of the individual hypotheses.

Figure 6.2 shows the performance of our algorithm and AdaBoost as a function of the average number of features accessed per example. On these problems, the SPEEDBOOST generated predictor finds a reasonable prediction using fewer features than the AdaBoost alternative and remains competitive with AdaBoost as time progresses.

6.5.2 Object Detection

Our second application is a vehicle detection problem using images from onboard cameras on a vehicle on public roads and highways under a variety of weather and time-of-day conditions. The positive class includes all vehicle types, e.g., cars, trucks, and vans. Negative examples are drawn from non-vehicle regions of images taken from the onboard cameras.

Prediction on Batch Data

In the previous application we consider weak predictors which will run for roughly the same amount of time on each example x and care about the performance of the learned predictor over time on a single example. In many settings, however, we often care about the computational requirements of a predictor on a batch of examples as a whole. For example, in ranking we care about the computation time required to get an accurate ranking on a set of items, and in computer vision applications many examples from a video or image are often processed simultaneously. Another way to view this problem is as a simplified version of the structured prediction problem where the goal is to make predictions on all pixels in an image simultaneously.

In these settings, it is often beneficial to allocate more computational resources to the difficult examples than the easy examples in a batch, so extra resources are not wasted improving predictions on examples that the algorithm already has high confidence in. In computer vision, in particular, cascades [Viola and Jones, 2001] are a popular approach to improving batch prediction performance. These prediction algorithms decrease the overall complexity of a predictor by periodically filtering out and making final predictions on examples, removing them from later prediction stages in the algorithm.

We can use our anytime framework and algorithms to consider running each weak predictor on subsets of the data instead of every example. Given a set of weak predictors $\mathcal{H}$ to optimize over, we can create a new set of predictors $\mathcal{H}'$ by introducing a set of filter functions $\phi \in \Phi$:

$$\phi : \mathcal{X} \to \{0, 1\},$$

and considering the pairing of every filter function and weak predictor

$$\mathcal{H}' = \Phi \times \mathcal{H}$$
$$h'(x) = \phi(x)h(x).$$

These filters ϕ represent the decision to either run the weak predictor h on example x or not. Unlike cascades, these decisions are not permanent and apply only to the current stage. This property very nicely allows the anytime predictor to quickly focus on difficult examples and gradually revisit the lower margin examples, whereas the cascade predictor must be highly-confident that an example is correct before halting prediction on that example.

Assuming that the filter function is relatively inexpensive to compute compared to the computation time of the predictor, the new complexity measure for predictors h' is

$$\tau(h') = \mathbb{E}_{\mathcal{X}} \left[\phi(x)\tau_x(h) \right],$$

or the expected computation time of the original predictor, only on image patches which are not filtered out by the filter function ϕ.

Implementation

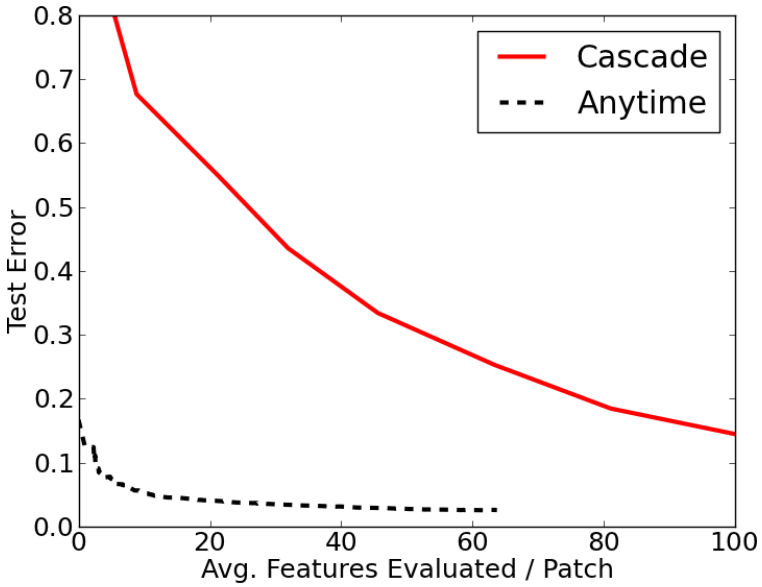


Figure 6.3: Test set error for the vehicle detection problem as a function of the average number of features evaluated on each image patch.

Similar to previous work in object detection, we use Haar-like features computed over image patches for weak predictors. We search over margin-based filter functions ϕ, such that the filters at stage i are

$$\phi_i(x) = \mathbb{1}(|f_{i-1}(x)| < \theta),$$

leveraging the property that examples far away from the margin are (with high probability) already correcly classified.

Computing these filters at test-time can be made relatively efficient in two ways. First, by storing examples in a priority queue sorted by current margin, the updates and filtering at each stage can be made relatively cheap. Second, after learning the anytime predictor using Algorithm 6.2, all future filters are known at each stage, and so the predictor can quickly determine the next stage an example will require computation in and handle the example accordingly.

We compare against a cascade implementation for this detection dataset which uses the standard AdaBoost algorithm for an inner loop learning algorithm. Figure 6.3 gives the error on a test dataset of 10000 positive and 50000 negative examples as a function of computation time. In this setting the cascade is at a significant disadvantage because it must solidly rule out any negative

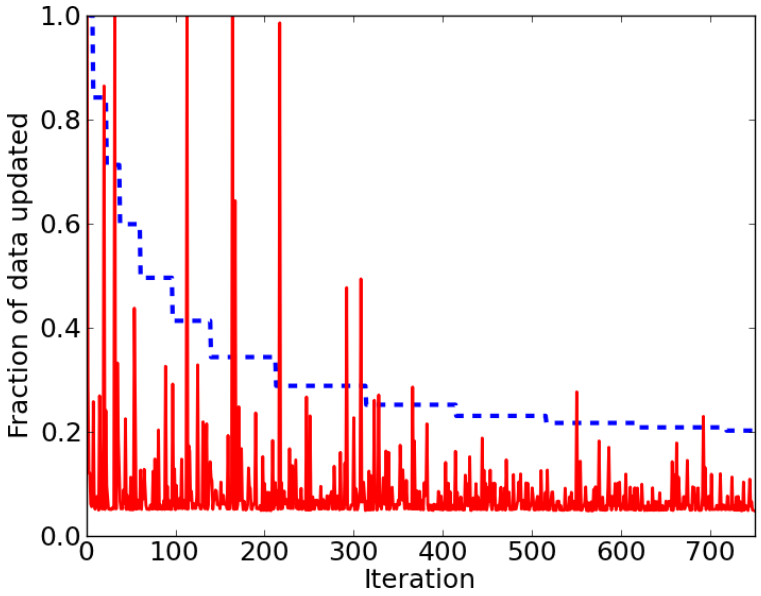


Figure 6.4: Fraction of data updated by each iteration for the cascade (dashed blue line) and anytime predictor (solid red line).

examples before classifying them as such, while the AdaBoost and anytime predictors can initially declare all examples negative and proceed to adjust prediction on positive examples.

To further illustrate the large benefit to being able to ignore examples early on and revisit them later, Figure 6.4 gives a per iteration plot of the fraction of test data updated by each corresponding feature. This demonstrates the large culling early on of examples that allows the anytime predictor to improve performance much more rapidly. Finally, Figure 6.5 displays the ROC curve for the anytime predictor at various complexity thresholds against the ROC curve generated by the final cascade predictions and Figure 6.6 shows the visual evolution of the cascade and anytime predictions on a single test image.

6.5.3 Budgeted Feature Selection

Our final application for the anytime prediction framework is in the feature selection domain. In this setting, we assume that the examples x do not have precomputed features, and that the total prediction time $\mathcal{T}$ is dominated by the computation time of the features of x required by our predictor. The goal, therefore, is to select a sequence of weak predictors h which only use a subset of the features for a given example x. At any given point we want to have selected the most efficient subset of features to obtain good anytime performance.

This is the same budgeted feature selection setting as Xu et al. [2012] and Xu et al. [2013a]. To handle this setting, we will have to slightly augment our cost model, to allow the cost of a weak predictor to be dependent on previously selected ones.

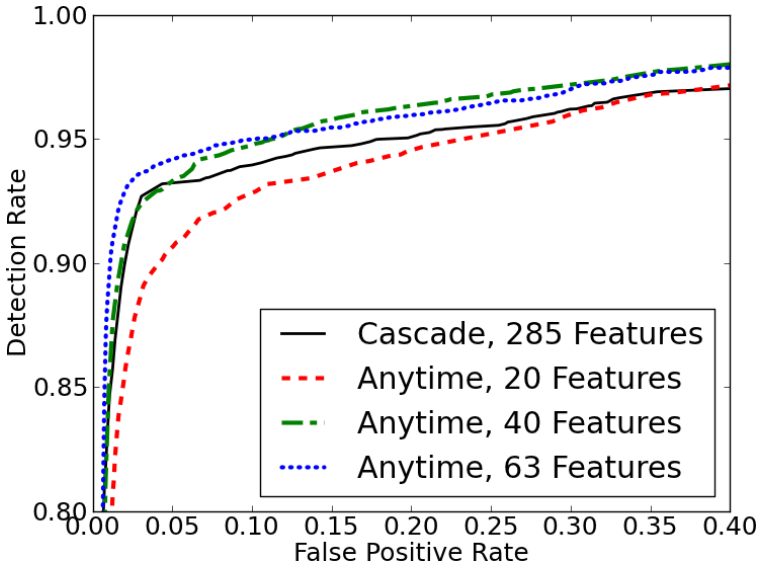


Figure 6.5: ROC curves for the final cascade predictions and anytime algorithm predictions at various computation thresholds. Computation is measured using the average number of features computed on each patch.

Dependent Weak Learner Costs

In the anytime prediction framework we presented in Section 6.2 we made the assumption that weak predictors have some fixed cost $\tau(h)$. However, in the budgeted feature selection model, features only incur computation costs the first time they are used. This implies that the cost of a new weak predictor is dependent on which features have been selected so far, and hence which predictors have been selected so far.

To handle this case, we now introduce a weak predictor cost which is conditioned on the previously selected weak predictors. When evaluating the cost of a weak predictor after selecting t weak predictors already, the cost would be

$$\tau(h|h_1, \ldots, h_t),$$

and the total predictor cost would be

$$\tau(f) = \sum_t \tau(h_t|h_1, \ldots, h_{t-1}).$$

In the budgeted feature selection setting we are given a set of features $\phi \in \Phi$ and we assume each feature has some computation or acquisition cost c_ϕ. We also assume that there is some small fixed cost for each predictor used, which represents the computational cost of evaluating that predictor, after all features have been computed. For example, this might be the cost of evaluating a decision tree once all features are known. We represent this fixed cost as c_h for predictor h.

Using the notation $\phi \in f$ to represent that feature ϕ is used by predictor f, we can write the conditional cost for the budgeted feature selection problem as

$$\tau(h|h_1, \ldots, h_t) = c_h + \sum_{\phi \in \Phi} c_\phi \mathbb{1}(\phi \in h) \prod_t \mathbb{1}(\phi \notin h_t).$$

As desired, this definition of cost only incurs a penalty for the first time a feature is computed. After that, the use of a feature is free, except for any fixed costs incurred in computing the predictor, captured by the cost c_h.

The total computation time of a predictor f then reduces exactly as we'd expect to

$$\tau(f) = \sum_t c_{h_t} + \sum_\phi c_\phi \mathbb{1}(\phi \in f).$$

To modify our anytime prediction algorithms for this setting we simply augment the greedy selection step in each algorithm to use the conditional cost instead of a fixed cost. This conditional cost requirement breaks the assumptions required for the theoretical guarantees in Section 6.4, but in practice the performance appears to be similar to the results we see when examining settings with fixed costs.

Cost-Regularized Regression Trees

In this domain we want to generate weak predictors h that incur a variety of costs, by using different mixtures of already computed and new features. Ideally, the weak predictors would also use new features with a variety of costs, to explore the possibilities of using cheap and expensive features.

To achieve this, we use the weak predictor proposed by Xu et al. [2012] in their Greedy Miser algorithm. Their weak predictor is based on a regression tree framework, but modifies the regression tree split function, also known as the impurity function, with a cost-based regularizer. Assume we are at iteration $t+1$, and have already learned a predictor f_t. The Greedy Miser weak predictor selects node splits using an impurity function g which optimizes the cost-regularized squared error

$$g(h) = \frac{1}{2} \sum_{n=1}^{N} \|\nabla(x_n) - h(x_n)\|^2 + \lambda \tau(h|f_t),$$

where λ is a regularization parameter which trades cost and accuracy of the learned predictor.

In the Greedy Miser approach, a fixed λ is chosen for all weak predictors, and functional gradient boosting proceeds as normal, using the cost-regularized weak learning algorithm. Using different values of λ produces different points on the cost and accuracy trade-off spectrum. In our approach, we will use SPEEDBOOST to optimize simultaneously over weak predictors learned with all the different values of λ. Specifically, we will use SPEEDBOOST.Proj (Algorithm 6.2) to generate the best regression tree for each value of λ in a pre-selected set of possible values, and then select from among these candidate trees using the cost-greedy SPEEDBOOST criteria.

Experiments

Our first problem in this domain is the Yahoo! Learning to Rank Challenge data, augmented with feature computation costs [Xu et al., 2012]. The dataset consists of a set of documents paired with relevance scores in $\{0, 1, 2, 3, 4\}$, with 0 representing a completely irrelevant result and a score of 4 indicating high relevance. The document, relevance pairs are then grouped by query, representing the groups within which the results are to be ranked. The dataset consists of 473134 training documents, along with 71083 and 165660 validation and testing documents.

The features for the data are drawn from a variety of sources, with costs drawn from the set $c_\phi \in \{1, 5, 10, 20, 50, 100, 150, 200\}$. We additionally use a fixed cost $c_h = 1$ for each tree in this problem.

For learning purposes we model the problem as a regression problem using squared error with the relevance scores for targets. In practice, the Normalized Discounted Cumulative Gain (NDCG) [Järvelin and Kekäläinen, 2002] is used to measure actual ranking performance, but we cannot optimize this metric directly.

We compare to the Greedy Miser approach [Xu et al., 2012] for a variety of values of $\lambda \in \{0, 0.5, 1, 2, 4, 10\}$. We use the same set of λ values and the regularized regression tree training detailed above to generate candidate weak predictors for the SPEEDBOOST.Proj algorithm. Additionally, for $\lambda = 0$, Greedy Miser is equivalent to the standard functional gradient approach. Although Greedy Miser is not an anytime approach per se, we can treat the predictor produced for a given fixed value of λ as an anytime predictor in the same way we treat the sequences generated by SPEEDBOOST.

Figure 6.7 gives the training performance, in the form of mean squared error, and test set accuracy , in the form of NDCG @ 5, for the Yahoo! Learning to Rank data. The anytime performance of each individual Greedy Miser sequence is plotted, along with the single sequence trained by SPEEDBOOST.

Looking at the training performance, we see that the SPEEDBOOST approach is very close to the optimal performance at any given computational budget with respect to the training objective of mean squared error. For test set performance, the NDCG @ 5 on test data is also nearly optimal, but here we observe a small increase in the overfitting of the cost-greedy SPEEDBOOST approach as compared with the Greedy Miser approach. We postulate that this is due to the cost-greedy algorithm's tendency to re-use features in order to get smaller gains, due to their very low cost as compared to computing new features. This repeated re-use of cheap features can lead to increased overfitting, particular for the regression trees used here.

The second application we consider is the Scene-15 scene categorization dataset. This dataset consists of 4485 images grouped in to 15 classes describing the contents of the scene. Example classes include: highway, office, forest and coast. Since this is a multiclass classification task, we use the softmax, or cross-entropy loss, which is the mutliclass generalization of the logistic loss.

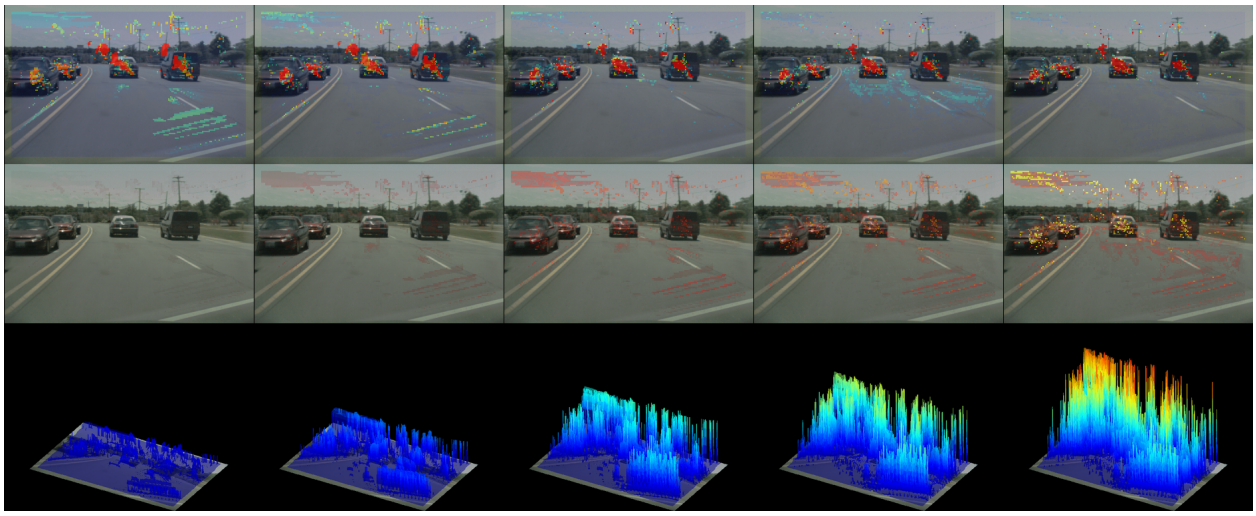
We utilize the same features and training procedure as Xu et al. [2012]. A total of 1500 images are sampled (100 from each class) and used as training data, 300 are used as validation and the remaining 2685 are used as test data. For each image, 184 different feature descriptors are com-

puted using a variety of methods, such as Local Binary Patterns and spatial HOG features. Each separate descriptor is then used to train 15 one-vs-all SVMs on 30% of the training data. Finally, the predictions of the trained SVMs are used as input features for the anytime learner, with the remaining 70% of the training data being used for training the anytime predictor. The end result is a set of 1050 training examples and $184 \times 15 = 2070$ input features for the functional gradient training procedure.

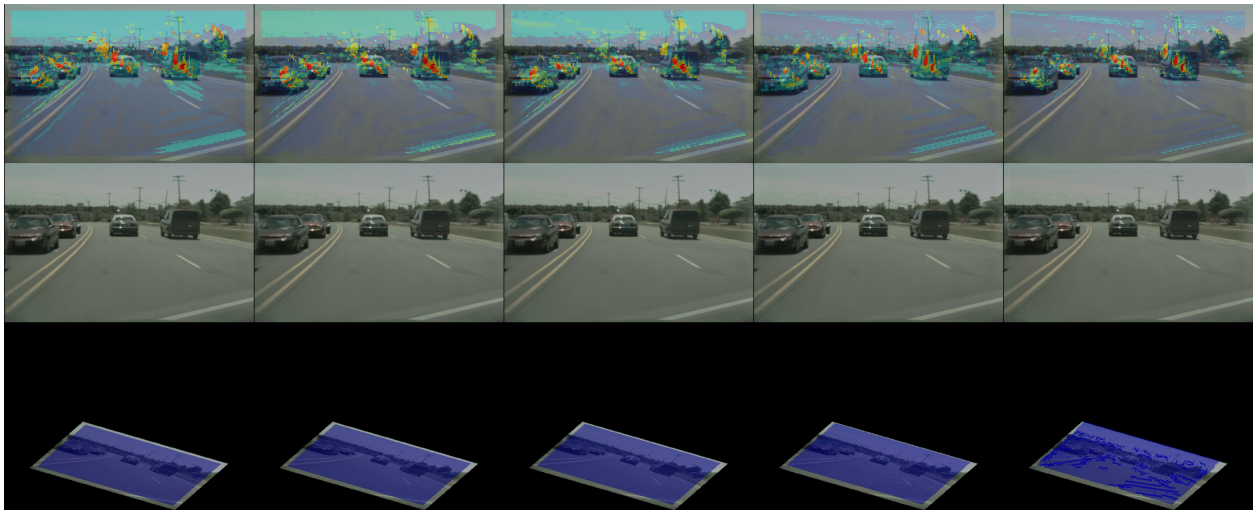
The feature costs in this case are derived from the cost of evaluating the underlying feature descriptor. Each feature corresponds to a specific one-vs-all SVM computed on one particular feature descriptor with a particular computational cost, represented as the time to compute that descriptor for the average image. In this particular case, because multiple SVMs are trained on each feature descriptor, we actually have costs for groups of features. That is, a weak predictor h only incurs cost for feature ϕ if no features ϕ' that are in the same group as ϕ have been computed yet. Computing a feature in a group makes all the other features have an effective cost of 0, because the computational cost is fixed for the entire descriptor and all features derived from it.

In this setting there are significantly more features than training examples (2070 vs. 1050). This makes it highly likely that the regression trees used as weak learners will overfit to the training data, even when using a small subset of the features. In practice we observe that, when using the standard SPEEDBOOST approach, the algorithm significantly overestimates the cost-greedy gain on training data. To increase robustness in this setting, we use a sampling approach similar to Stochastic Gradient Boosting [Friedman, 1999]. At each iteration, we sample 90% of the training data without replacement and use this training data for gradient projection, *i.e.* training weak predictors. We then evaluate the cost-greedy gain used to select the optimal weak predictor in SPEEDBOOST on the remaining 10% of the data that was held out. By evaluating the cost-greedy gain on held out data, we compute an estimate of the true cost-greedy gain that is much closer to the behavior on test data.

Figure 6.8 gives the same comparison of training objective and test accuracy for different computational budgets on the Scene-15 dataset. For this dataset we see similar behavior as the Yahoo LTR application. Though there are certain budgets for which the best fixed Greedy Miser predictor outperforms the SPEEDBOOST predictor, overall the SPEEDBOOST approach is nearly as good as the best performing predictor for a given budget. Furthermore, in this setting the fixed predictors show very little change in features selected over time and largely target a single set of features and hence a single budget. The SPEEDBOOST predictor, in constrast, initially uses cheaper features and then switches to expensive features when doing so maximally increases the gain. Also note that, while the training objective can continue to be decreased using only cheap features, as the Greedy Miser predictor for $\lambda = 4$ shows, the sampling strategy ensures that the predictor switches to using expensive features when doing so is beneficial on validation data.



(a)



(b)

Figure 6.6: Images displaying the performance on a test image for the anytime predictor produced by SPEEDBOOST (top) and the cascade (bottom). Displayed on the top of each image are the activations, or classification probabilities, for that algorithm. In the middle is a heat map of the number of features evaluated by that predictor for each pixel, along with a 3D visualization of this same statistic along the bottom. Images are arranged left to right through time, at intervals of 7 average feature evaluations per pixel. Note that, at this scale, the cascade still has most of its effort spread over the entire image, and so the heatmap and 3D visualization are largely flat.

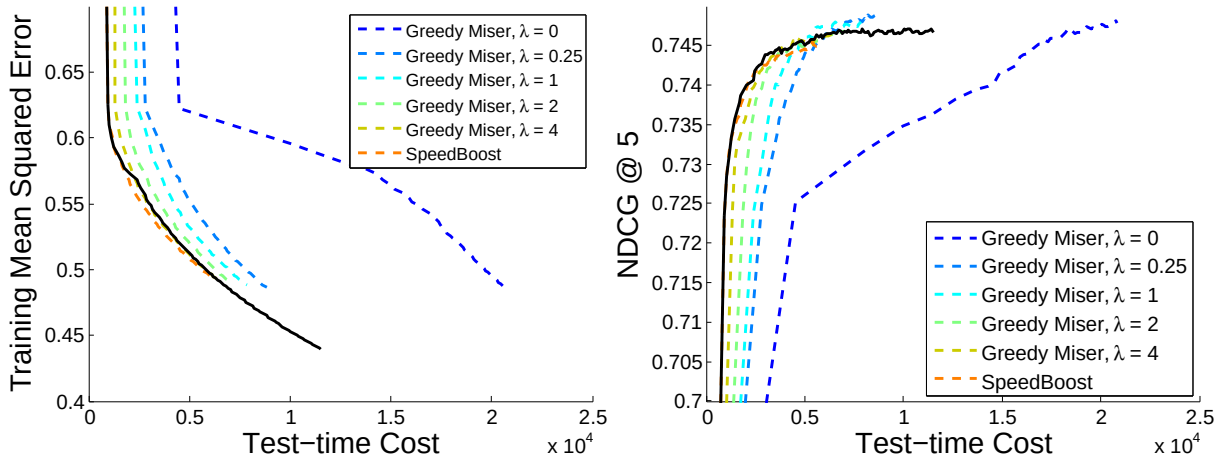


Figure 6.7: Training objective (left) and test set accuracy (right) vs. computational cost for the budgeted version Yahoo! Learning to Rank Challenge problem. Provided for comparison are the SPEEDBOOST.Proj algorithm along with Greedy Miser [Xu et al., 2012] for a variety of regularization parameters λ. For $\lambda = 0$, the Greedy Miser approach is equivalent to standard functional gradient boosting.

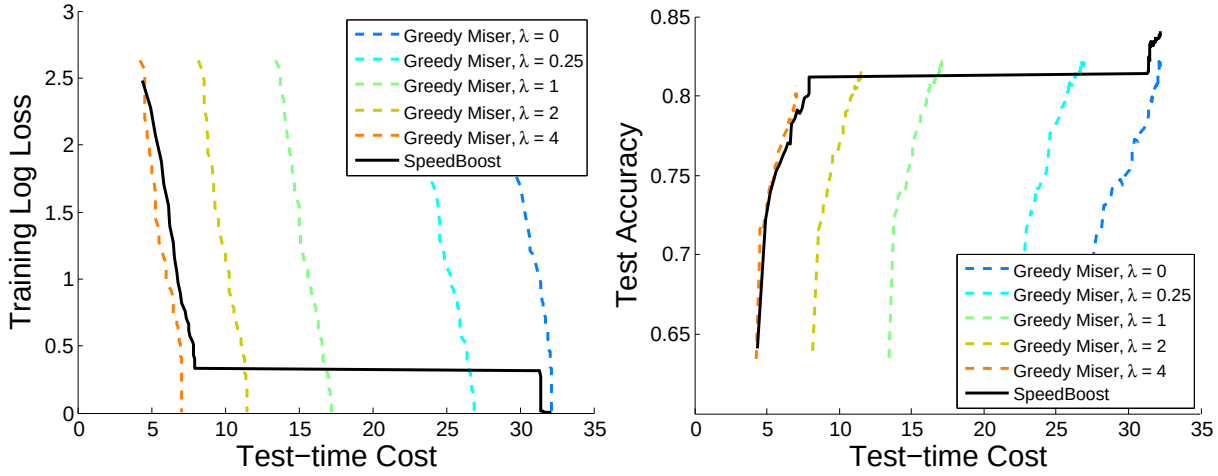


Figure 6.8: Training objective (left) and test set accuracy (right) vs. computational cost for the Scene-15 scene categorization dataset. Provided for comparison are the SPEEDBOOST.Proj algorithm along with Greedy Miser [Xu et al., 2012] for a variety of regularization parameters λ. For $\lambda = 0$, the Greedy Miser approach is equivalent to standard functional gradient boosting.

Chapter 7

STRUCTUREDSPEEDBOOST: Anytime Structured Prediction

In this chapter we will demonstrate another application of our anytime predicition framework to the structured prediction setting, specifically to the scene understanding domain. To do so, we will combined the cost-greedy SPEEDBOOST approach detailed in the previous chapter with the structured prediction extensions of functional gradient methods which we detailed in Chapter 3.

7.1 Background

We will first briefly review the structured functional gradient previously detailed in Chapter 3, specifically Section 3.1. For more details on the structured prediction approach, we refer the reader to that chapter.

Recall the structured prediction setting previously discussed in Section 3.1. In this setting we are given some inputs $x \in \mathcal{X}$ and associated structured outputs $y \in \mathcal{Y}$. The goal is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes some risk $\mathcal{R}[f]$, typically evaluated pointwise over the inputs:

$$\mathcal{R}[f] = \mathbb{E}_{\mathcal{X}}[\ell(f(x))]. \tag{7.1}$$

We also assume the structured outputs are representable as a variable length vector $(y_1, \ldots, y_J)$, where the output y_j represents the output for some structural element of the total output y. For example, the structural elements may be the probability distribution over class labels for a pixel in an image or the current prediction for a node in a graphical model.

We also assume that each output j has associated with it some set $N(j)$ which represents the locally connected elements of the structure of the problem, such as the locally connected factors of a node j in a typical graphical model. For a given node j, the predictions over the neighboring nodes $\hat{y}_{N(j)}$ and other features of the local structure $N(j)$ can then be used to update the prediction for the node j, in a manner similar to the message passing approach commonly used for graphical model prediction [Pearl, 1988].

As we did before, to approach such a structured prediction problem we will be using an additive, functional gradient version of the iterative decoding approach [Cohen and Carvalho, 2005, Daume III et al., 2009, Tu and Bai, 2010, Socher et al., 2011, Ross et al., 2011]. In this approach, we are going to learn an additive structured predictor,

$$h(x, \hat{y}^t)_j = \mathbb{1}(j \in h_{\mathrm{S}}(x, \hat{y}^t)) h_{\mathrm{P}}(x_j, \hat{y}^t_{N(j)}), \tag{7.2}$$

that consists of two main components. The first, a selection function h_{S}, which selects some subset of the structural elements to update at each iteration, and a predictor h_{P} which runs on the selected elements and updates the respective pieces of the structured output.

To complete the structured prediction extension, we need a method for selecting weak predictors h as specified in Equation (7.2). Following the functional gradient approach detailed in Chapter 2, and extended to structured prediction setting as in Section 3.1, we will use projected functional gradients for this purpose.

There is typically no efficient way to train a selection function and predictor simultaneously, so we will instead choose selector, predictor pairs by first enumerating selection functions h_{S} and then using functional gradient methods to select the optimal h_{P} for the chosen selector.

We can compute a functional gradient with respect to each element of the structured output,

$$\nabla(x)_j = \frac{\partial \ell(f(x))}{\partial f(x)_j}.$$

Given a fixed selection function h_{S} and current predictions $\hat{y}$, the functional gradient projection for finding the optimal weak predictors h_{P} is as follows. In order to minimize the projection error in Equation (2.12) for a predictor h of the form in Equation (7.2), we only need to find the prediction function h_{P} that minimizes

$$h^*_{\mathrm{P}} = \operatorname*{arg\,min}_{h_{\mathrm{P}} \in \mathcal{H}_{\mathrm{P}}} \mathbb{E}_{\mathcal{X}} \left[\sum_{j \in h_{\mathrm{S}}(x, \hat{y})} \left\| \nabla(x)_j - h_{\mathrm{P}}(x_j, \hat{y}_{N(j)}) \right\|^2 \right]. \tag{7.3}$$

This optimization problem is equivalent to minimizing weighted least squares error over the dataset

$$\begin{aligned} D &= \bigcup_x \bigcup_{j \in h_{\mathrm{S}}(x, \hat{y})} \{(\psi_j, \nabla(x)_j)\}, \\ &= \text{gradient}(f, h_{\mathrm{S}}), \end{aligned} \tag{7.4}$$

where $\psi_j = \psi(x_j, \hat{y}_{N(j)})$ is a feature descriptor for the given structural node, and $\nabla(x)_j$ is its target. In order to model contextual information, ψ is drawn from both the raw features x_j for the given element and the previous locally neighboring predictions $\hat{y}_{N(j)}$.

The functional gradient algorithm for learning these additive structured predictors was given previously in Algorithm 3.1.

7.2 Anytime Structured Prediction

We now combine the structured functional gradient methods developed in Chapter 3 with the anytime prediction techniques developed in Chapter 6.

Recall that in the anytime setting we have a cost $c(h)$ for each weak predictor h, and that the SPEEDBOOST approach (Section 6.3) we use a cost-greedy criteria for selecting predictors h:

$$h_t, \alpha_t = \arg\max_{h \in \mathcal{H}, \alpha \in \mathbb{R}} \frac{\mathcal{R}\left[f_{t-1}\right] - \mathcal{R}\left[f_{t-1} + \alpha h\right]}{c(h)}. \tag{7.5}$$

The adapted cost model for the additive weak predictor (Equation (7.2)) is then simply the sum of the cost of evaluating both the selection function and the prediction function,

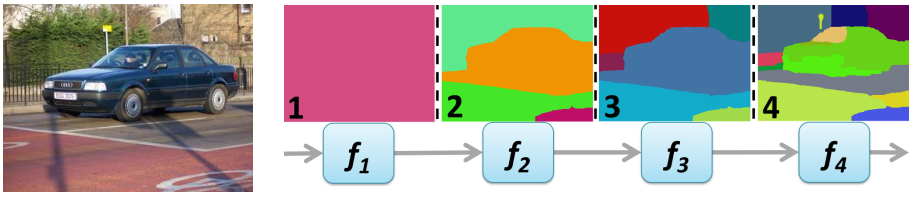
$$c(h) = c(h_{\text{S}}) + c(h_{\text{P}}). \tag{7.6}$$

Algorithm 7.1 summarizes the STRUCTUREDSPEEDBOOST algorithm for anytime structured prediction. It is based off of the structured functional gradient algorithm (Algorithm 3.1), modified with the cost-greedy SPEEDBOOST criteria from Chapter 6 to select the most cost efficient pair of selection and prediction functions.

It enumerates the candidate selection functions, h_{S}, creates the training dataset defined by Equation (7.4), and then generates a candidate prediction function h_{P} using each weak learning algorithm. For all the pairs of candidates, it uses Equation (7.5) for picking the best pair, instead of the non-anytime version, which simply optimizes the regular functional gradient criteria.

Algorithm 7.1 STRUCTUREDSPEEDBOOST

Given: objective $\mathcal{R}$, set of selection functions $\mathcal{H}_{\text{S}}$, set of L learning algorithms $\{\mathcal{A}_l\}_{l=1}^{L}$, number of iterations T, initial function f_0.
for $t = 1, \ldots, T$ **do**
  $\mathcal{H}^* = \emptyset$
  for $h_{\text{S}} \in \mathcal{H}_{\text{S}}$ **do**
    Create dataset $D = \text{gradient}(f_{t-1}, h_{\text{S}})$ using Equation (3.9).
    for $\mathcal{A} \in \{\mathcal{A}_1, \ldots, \mathcal{A}_L\}$ **do**
      Train $h_{\text{P}} = \mathcal{A}(D)$
      Define h from h_{S} and h_{P} using Equation (3.6).
      $\mathcal{H}^* = \mathcal{H}^* \cup \{h\}$
    end for
  end for
  $h_t, \alpha_t = \arg\max_{h \in \mathcal{H}^*, \alpha \in \mathbb{R}} \frac{\mathcal{R}[f_{t-1}] - \mathcal{R}[f_{t-1} + \alpha h]}{\tau(h)}$
  $f_t = f_{t-1} + \alpha_t h_t$
end for



(a) (b)

Figure 7.1: Hierarchical Inference Machines [Munoz et al., 2010]. (a) Input image. (b) The image is segmented multiple times; predictions are made and passed between levels. Images courtesy of the authors' ECCV 2010 presentation.

7.3 Anytime Scene Understanding

7.3.1 Background

In addition to part-of-speech tagging in natural language processing, scene understanding in computer vision is another important and challenging structured prediction problem. The *de facto* approach to this problem is with random field based models [Kumar and Hebert, 2006, Gould et al., 2008, Ladicky et al., 2010], where the random variables in the graph represent the object category for a region/patch in the image. While random fields provide a clean interface between modeling and inference, recent works [Tu and Bai, 2010, Munoz et al., 2010, Socher et al., 2011, Farabet et al., 2013] have demonstrated alternative approaches that achieve equivalent or improved performances with the additional benefit of a simple, efficient, and modular inference procedure.

Inspired by the hierarchical representation used in the state-of-the-art scene understanding technique from Munoz et al. [2010], we apply STRUCTUREDSPEEDBOOST to the scene understanding problem by reasoning over differently sized regions in the scene. In the following, we briefly review the hierarchical inference machine (HIM) approach from [Munoz et al., 2010] and then describe how we can perform an anytime prediction whose structure is similar in spirit.

7.3.2 Hierarchical Inference Machines

HIM parses the scene using a hierarchy of segmentations, as illustrated in Figure 7.1. By incorporating multiple different segmentations, this representation addresses the problem of scale ambiguity in images. Instead of performing (approximate) inference on a large random field defined over the regions, inference is broken down into a sequence of predictions. As illustrated in Figure 7.1, a predictor f is associated with each level in the hierarchy that predicts the probability distribution of classes/objects contained within each region. These predictions are then used by the subsequent predictor in the next level (in addition to features derived from the image statistics) to make refined predictions on the finer regions; and the process iterates. By passing class distributions between predictors, contextual information is modeled even though the segmentation at any particular level may be incorrect. We note that while Figure 7.1 illustrates a top-down sequence

over the hierarchy, in practice, the authors iterate up and down the hierarchy which we also do in our comparison experiments.

7.3.3 Speedy Inference Machines

While HIM decomposes the structured prediction problem into an efficient sequence of predictions, it is not readily suited for an anytime prediction. **First**, the final predictions are generated when the procedure terminates at the leaf nodes in the hierarchy. Hence, interrupting the procedure before then would result in final predictions over coarse regions that may severely undersegment the scene. **Second**, the amount of computation time at each step of the procedure is invariant to the current performance. Because the structure of the sequence is predefined, the inference procedure will predict multiple times on a region as it traverses over the hierarchy, even though there may be no room for improvement. **Third**, the input to each predictor in the sequence is a fixed feature descriptor for the region. Because these input descriptors must be precomputed for all regions in the hierarchy before the inference process begins, there is a fixed initial computational cost. In the following, we describe how STRUCTUREDSPEEDBOOST addresses these three problems three problems for anytime scene understanding.

Scene Understanding Objective

In order to address the first issue, we learn an additive predictor f which predicts a per-pixel classification for the entire image at once. In contrast to HIM whose *multiple* predictors' losses are measured over regions, we train a *single* predictor whose loss is measured over pixels. Concretely, given per-pixel ground truth distributions $p_j \in \mathbb{R}^K$, we wish to optimize per-pixel, cross-entropy risk for all pixels in the image

$$\mathcal{R}[f] = \mathbb{E}_{\mathcal{X}} \left[-\sum_j \sum_k p_{jk} \log q(f(x))_{jk} \right], \tag{7.7}$$

where

$$q(y)_{jk} = \frac{\exp(y_{jk})}{\sum_{k'} \exp(y_{jk'})}, \tag{7.8}$$

i.e., the probability of the k'th class for the j'th pixel. Using Equation (7.2), the probability distribution associated with each pixel is then dependent on 1) the pixels to update, selected by h_{S}, and 2) the value of the predictor h_{P} evaluated on those respective pixels. The definition of these functions are defined in the following subsections.

Structure Selection and Prediction

In order to account for scale ambiguity and structure in the scene, we can similarly integrate multiple regions into our predictor. By using a hierarchical segmentation of the scene that produces

many segments/regions, we can consider each resulting region or segment of pixels S in the hierarchy as one possible set of outputs to update. Intuitively, there is no need to update regions of the image where the predictions are correct at the current inference step. Hence, we want to update the portion of the scene where the predictions are uncertain, *i.e.*, have high entropy H. To achieve this, we use a selector function that selects regions that have high average per-pixel entropies in the current predictions,

$$h_{\mathrm{S}}(x, \hat{y}) = \left\{ S \;\middle|\; \frac{1}{|S|} \sum_{j \in S} H(q(\hat{y})_j) > \theta \right\}, \tag{7.9}$$

for some fixed threshold θ. In practice, the set of predictors $\mathcal{H}_{\mathrm{S}}$ used at training time is created from a diverse set of thresholds θ.

Additionally, we assume that the features ψ_j used for each pixel in a given selected region are drawn from the entire region, so that if a given scale is selected features corresponding to that scale are used to update the selected pixels. For a given segment S, call this feature vector ψ_S.

Given the above selector function, we use Equation (7.3) to find the next best predictor function, as in Algorithm 7.1, optimizing

$$h_{\mathrm{P}}^* = \arg\min_{h_{\mathrm{P}}} \sum_{S \in h_{\mathrm{S}}(x,\hat{y})} \sum_{j \in S} \left\| \nabla(x)_j - h_{\mathrm{P}}(\psi_S) \right\|^2. \tag{7.10}$$

Because all pixels in a given region use the same feature vector, this reduces to the weighted least squares problem:

$$h_{\mathrm{P}}^* = \arg\min_{h_{\mathrm{P}}} \sum_{S \in h_{\mathrm{S}}(x,\hat{y})} |S| \left\| \nabla_S - h_{\mathrm{P}}(\psi_S) \right\|^2. \tag{7.11}$$

where $\nabla_S = \mathbb{E}_{j \in S}[\nabla(x)_j] = \mathbb{E}_{j \in S}[p_j - q(\hat{y})_j]$. In words, we find a vector-valued regressor h_{P} with minimal weighted least squares error between the difference in ground truth and predicted per-pixel distributions, averaged over each selected region/segment, and weighted by the size of the selected region. This is an intuitive update that places large weight to updating large regions.

Dynamic Feature Computation

In the scene understanding problem, a significant computational cost during inference is often feature descriptor computation. To this end, we utilize the SPEEDBOOST cost model Equation (7.6) to automatically select the most computationally efficient features.

The features used in this application, drawn from previous work [Gould et al., 2008, Ladicky, 2011] and detailed in the following section, are computed as follows. First, a set of base feature descriptors are computed from the input image data. In many applications it is useful to quantize these base feature descriptors and pool them together to form a set of derived features [Coates

FH	SHAPE	TXT (B)	TXT (D)	LBP (B)	LBP (D)	I-SIFT (B)	I-SIFT (D)	C-SIFT (B)	C-SIFT (D)
167	2	29	66	64	265	33	165	93	443

Table 7.1: Average timings (ms) for computing all features for an image in the SBD. (B) is the time to compute the *base* per-pixel feature responses, and (D) is the time to compute the *derived* averaged-pooled region codes to all cluster centers.

et al., 2011]. We follow the soft vector quantization approach in [Coates et al., 2011] to form a quantized code vector by computing distances to multiple cluster centers in a dictionary.

This computation incurs a fixed cost for 1) each group of features with a common base feature, and 2) an additional, smaller fixed cost for each actual feature used. In order to account for these costs, we use an additive model similar to Xu et al. [2012] and the budgeted feature selection application examine in the experimental analysis of the SPEEDBOOST algorithm, in Section 6.5.3.

Formally, let $\phi \in \Phi$ be the set of features and $\gamma \in \Gamma$ be the set of feature groups, and c_ϕ and c_γ be the cost for computing derived feature ϕ and the base feature for group γ, respectively. Let $\Phi(f)$ be the set of features used by predictor f and $\Gamma(f)$ the set of its used groups. Given a current predictor f_{t-1}, its group and derived feature costs are then just the costs of any new group and derived features and have not previously been computed:

$$c_\Gamma(h_{\mathrm{P}}) = \sum_{\gamma \in \Gamma(h_{\mathrm{P}}) \setminus \Gamma(f_{t-1})} c_\gamma,$$
$$c_\Phi(h_{\mathrm{P}}) = \sum_{\phi \in \Phi(h_{\mathrm{P}}) \setminus \Phi(f_{t-1})} c_\phi.$$

The total cost model in Equation (7.6) can then be derived using the sum of the feature costs and group costs as

$$\begin{aligned} c(h) &= c(h_{\mathrm{s}}) + c(h_{\mathrm{P}}) \\ &= \epsilon_{\mathrm{S}} + \epsilon_{\mathrm{P}} + c_\Gamma(h_{\mathrm{P}}) + c_\Phi(h_{\mathrm{P}}), \end{aligned} \tag{7.12}$$

where ϵ_{S} and ϵ_{P} are small fixed costs for evaluating a selection and prediction function, respectively.

In order to generate h_{P} with a variety of costs, we use a modified regression tree that penalizes each split based on its potential cost, as in [Xu et al., 2012]. This approach augments the least-squares regression tree impurity function with a cost regularizer:

$$\mathbb{E}_D \left[w_D \| y_D - h_{\mathrm{P}}(x_D) \|^2 \right] + \lambda \left(c_\Gamma(h_{\mathrm{P}}) + c_\Phi(h_{\mathrm{P}}) \right), \tag{7.13}$$

where λ regularizes the cost. In addition to Equation (7.12), training regression trees with different values of λ, enables STRUCTUREDSPEEDBOOST to automatically select the most cost-efficient predictor.

	Sky	Tree	Road	Grass	Water	Bldg.	Mtn.	Object	*Class*	*Pixel*
SIM	92.4	73.3	90.6	83.0	62.5	76.9	10.5	63.8	69.1	78.8
HIM [Munoz et al., 2010]	92.9	77.5	89.9	83.6	70.9	83.2	17.2	69.3	73.1	82.1
[Farabet et al., 2013]	95.7	78.7	88.1	89.7	68.7	79.9	44.6	62.3	76.0	81.4
[Socher et al., 2011]	-	-	-	-	-	-	-	-	-	78.1

	Bldg.	Tree	Sky	Car	Sign	Road	Ped.	Fence	Pole	Sdwlk.	Bike	*Class*	*Pixel*
SIM	76.8	79.2	94.1	71.8	30.2	95.2	34.6	25.7	14.0	66.3	15.0	54.8	81.5
HIM [Munoz et al., 2010]	83.3	82.2	95.9	75.2	42.2	96.0	38.6	21.5	13.6	72.1	33.3	59.4	84.9
[de Nijs et al., 2012]	59	75	93	84	45	90	53	27	0	55	21	54.7	75.0
[Ladicky et al., 2010]†	81.5	76.6	96.2	78.7	40.2	93.9	43.0	47.6	14.3	81.5	33.9	62.5	83.8

Table 7.2: Recalls on the Stanford Background Dataset (top) and CamVid (bottom) where *Class* is the average per-class recall and *Pixel* is the per-pixel accuracy. †Uses additional training data not leveraged by other techniques.

7.4 Experimental Analysis

7.4.1 Setup

We evaluate performance metrics between SIM and HIM on the 1) Stanford Background Dataset (SBD) [Gould et al., 2009], which contains 8 classes, and 2) Cambridge Video Dataset (CamVid) [Brostow et al., 2008], which contains 11 classes; we follow the same training/testing evaluation procedures as originally described in the respective papers. As shown in Table 7.2, we note that HIM achieves state-of-the-art performance and these datasets and analyze the computational tradeoffs when compared with SIM. Since both methods operate over a region hierarchy of the scene, we use the same segmentations, features, and regression trees (weak predictors) for a fair comparison.

Segmentations

We construct a 7-level segmentation hierarchy by recursively executing the graph-based segmentation algorithm (`FH`) [Felzenszwalb and Huttenlocher, 2004] with parameters

$$\sigma = 0.25, c = 10^2 \times [1, 2, 5, 10, 50, 200, 500],$$
$$k = [30, 50, 50, 100, 100, 200, 300].$$

These values were qualitatively chosen to generate regions at different resolutions.

Features

A region's feature descriptor is composed of 5 feature groups (Γ): 1) region boundary shape/geometry/location (SHAPE) [Gould et al., 2008], 2) texture (TXT), 3) local binary patterns (LBP), 4) SIFT over intensity (I−SIFT), 5) SIFT separately over colors R, G, and B (C−SIFT). The last 4 are derived from per-pixel descriptors for which we use the publicly available implementation from [Ladicky, 2011].

Computations for segmentation and features are shown in Table 7.1; all times were computed on an Intel i7-2960XM processor. The SHAPE descriptor is computed solely from the segmentation boundaries and is efficient to compute. The remaining 4 feature group computations are broken down into the per-pixel descriptor (base) and the average-pooled vector quantized codes (derived), where each of the 4 groups are quantized separately with a dictionary size of 150 elements/centers using k-means. For a given pixel descriptor, v, its code assignment to cluster center, μ_i, is derived from its squared L_2 distance $d_i(v) = \|v - \mu_i\|_2^2$. Using the soft code assignment from [Coates et al., 2011], the code is defined as $\max(0, z_i(v))$, where

$$z_i(v) = \mathbb{E}_j[d_j(v)] - d_i(v) \tag{7.14}$$

$$= \mathbb{E}_j[\|\mu_j\|^2] - 2\langle \mathbb{E}_j[\mu_j], v\rangle - (\|\mu_i\|^2 - 2\langle \mu_i, v\rangle). \tag{7.15}$$

Note that the expectations are indepndent from the query descriptor v, hence the i'th code can be computed independently and enables selective computation for the region. The resulting quantized pixel codes are then averaged within each region. Thus, the costs to use these derived features are dependent if the pixel descriptor has already been computed or not. For example, when the weak learner *first* uses codes from the I−SIFT group, the cost incurred is the time to compute the I−SIFT pixel descriptor *plus* the time to compute distances to each specified center.

7.4.2 Analysis

In Figure 7.4 we show which cluster centers, from each of the four groups, are being selected by SIM as the inference time increases. We note that efficient SHAPE descriptor is chosen on the first iteration, followed by the next cheapest descriptors TXT and I−SIFT. Although LBP is cheaper than C−SIFT, the algorithm ignored LBP because it did not improve prediction wrt cost.

In Figure 7.2, we compare the classification performance of SIM and several other algorithms with respect to inference time. We consider HIM as well as two variants which use a limited set of the 4 feature groups (only TXT and TXT & I−SIFT); these SIM and HIM models were executed on the same computer. We also compare to the reported performances of other techniques and stress that these timings are reported from different computing configurations. The single anytime predictor generated by our anytime structured prediction approach is competitive with all of the specially trained, standalone models without requiring any of the manual analysis necessary to create the different fixed models.

In Figure 7.3, we show the progress of the SIM algorithm as it processes a scene from each of the datasets. Over time, we see the different structural nodes (regions) selected by the algorithm as well as improving classification.

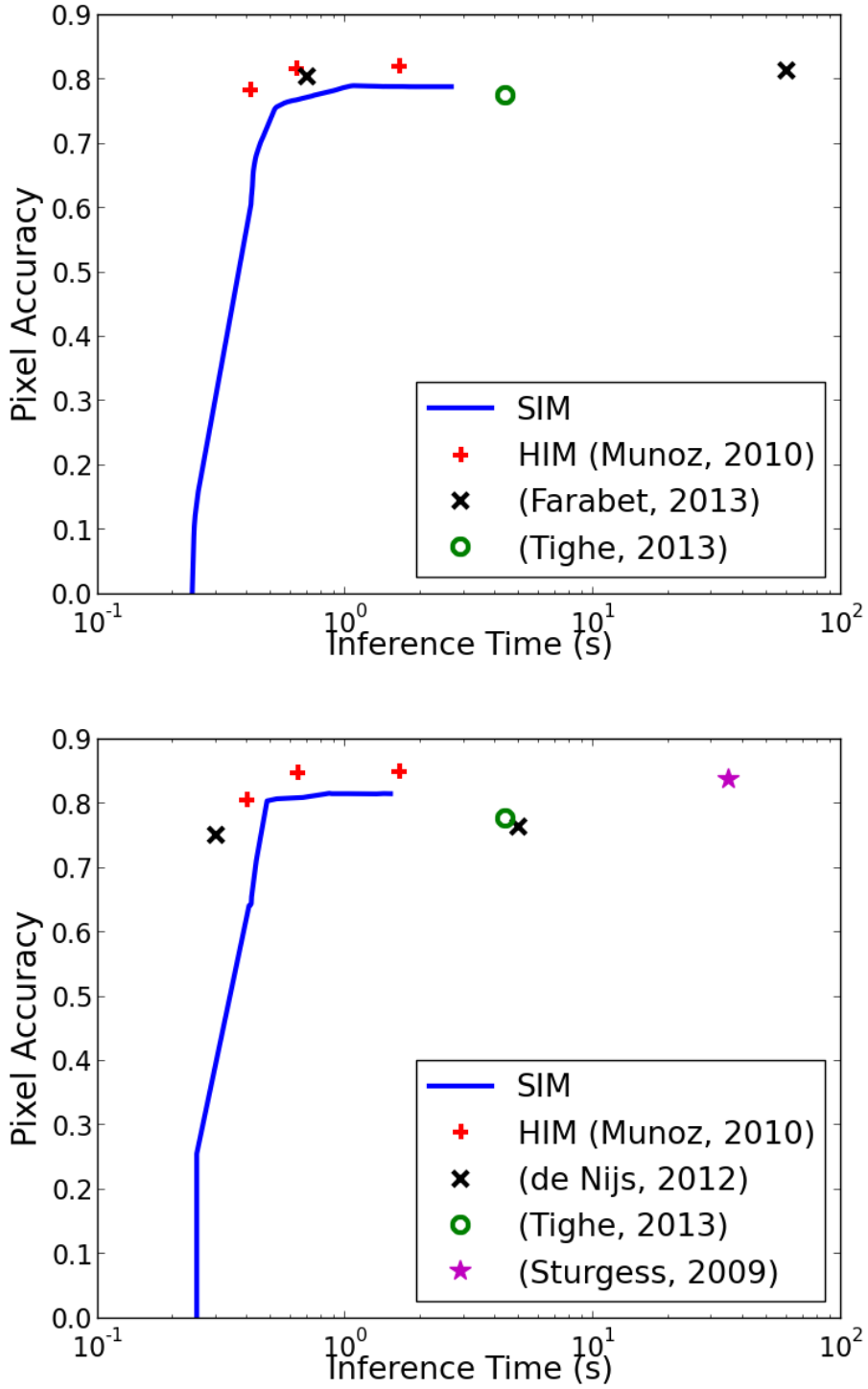


Figure 7.2: Average pixel classification accuracy for SBD (top) and CamVid (bottom) datasets as a function of inference time.

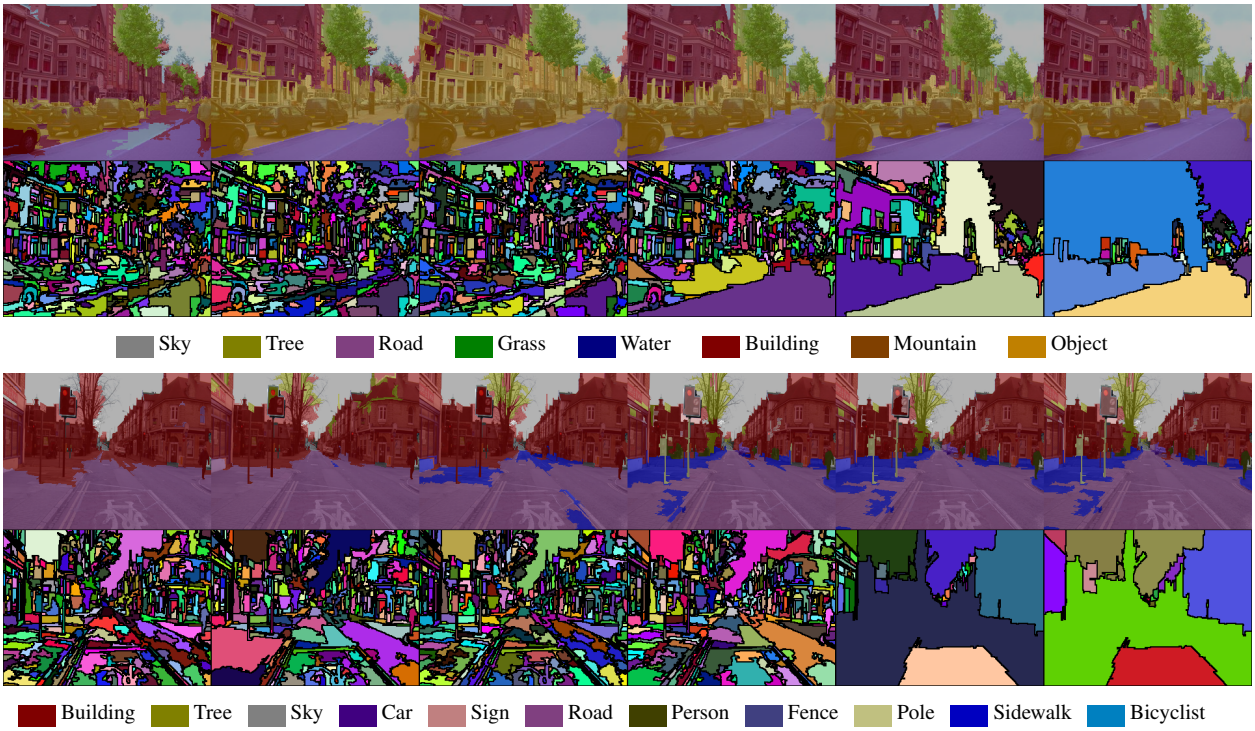


Figure 7.3: Sequence of images displaying the inferred labels and selected regions at iterations $t = \{1, 5, 15, 50, 100, 225\}$ of the SIM algorithm for a sample image from the Stanford Background (top) and CamVid (bottom) datasets. The corresponding inference times for these iterations are $\{0.42s, 0.44s, 0.47s, 0.79s, 1.07s, 1.63s\}$ (top) and $\{0.41s, 0.42s, 0.44s, 0.52s, 0.85s, 1.42s\}$ (bottom).

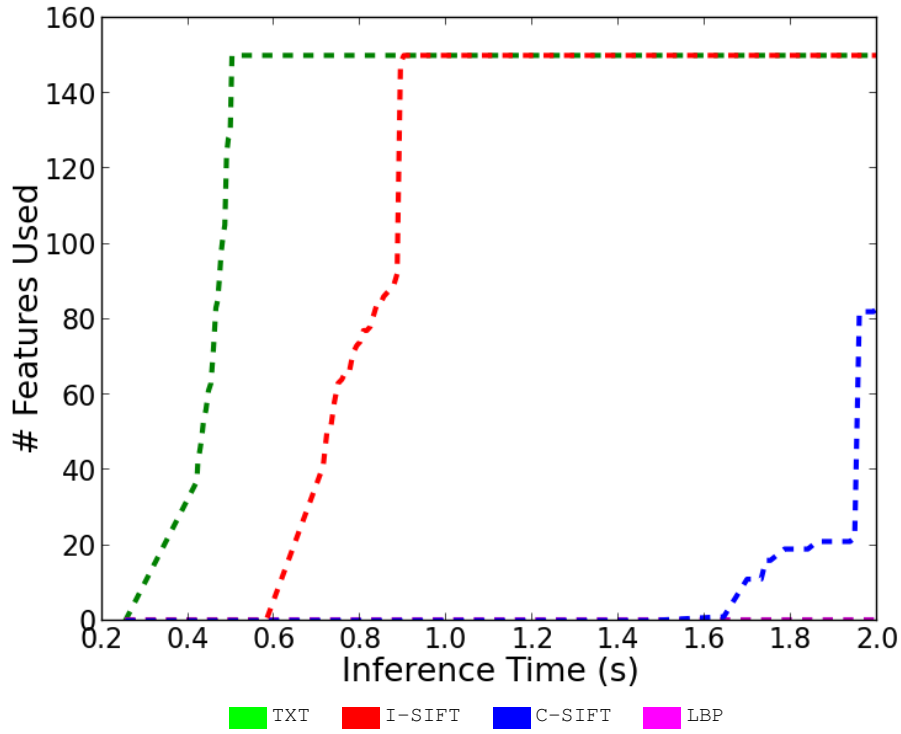


Figure 7.4: The number of cluster centers selected within each feature group by SIM as a function of inference time.

Chapter 8

Conclusion

8.1 Future Directions

In this thesis we propose a framework for anytime prediction and give algorithms for learning anytime predictors based on functional gradient methods and greedy optimization. Using these two areas, we give analysis and theoretical guarantees that show that our anytime prediction algorithms make near-optimal trade-offs between cost and accuracy without knowing the prediction time constraints apriori. There are a number of areas where we believe this sequential anytime prediction approach and accompanying analysis can be extended.

8.1.1 Anytime Representation Learning

Xu et al. [2013a] have proposed an anytime prediction approach which learns *feature representations* which change over time, and then computes predictions using these representations. In a similar fashion to deep network approaches, they learn a set of predictors $\{f_d\}_{d=1}^{D}$ which output a D dimensional feature representation and then combine this representation with a top layer using some simple linear function of the learned features, such as a Support Vector Machine.

We imagine that a similar approach to anytime representation learning could be derived using SPEEDBOOST as a base along with our prior work in backpropagated functional gradient techniques [Grubb and Bagnell, 2010] to build a similar network. By learning the layer of representation predictors f_d using a cost-greedy SPEEDBOOST approach and using functional gradient backpropagation to optimize the complete network, we may be able to improve the cost, accuracy trade-off behavior of their previous anytime representation learning approach, or even present a simpler algorithm for learning anytime representation learners.

It would be interesting to see how this approach compares to the previous work in anytime representation learning, and if this anytime representation approach offers any advantages over the direct anytime prediction approach we've learned here. Perhaps the representation learned could be used to enable anytime behavior in some other setting by using the anytime representation as

input, such as anytime clustering or anytime dimensionality reduction.

8.1.2 Parallel Weak Predictors

In this document, we have considered only the computational model where weak predictors are sequentially applied to a given input until interrupted. One alternative model is to consider the setting where weak predictors can freely be run in parallel. Consider the example of a large-scale web service which must compute predictions on a large number of inputs simultaneously. Typically in such a setting there are a number of other prediction algorithms running remotely that provide weak predictions to the overall web-service. In this case selecting a single weak predictor to run is counterproductive, as it is much more efficient to make requests to the other services in parallel.

In the parallel setting, we imagine that there are actually two computational constraints at play: the amount of parallelism available and the amount of time or total computation available to the system. One approach would be to only consider weak predictors that are parallelized to take advantage of any parallel resources, and revert to the sequential model we've outlined here.

An approach that would perhaps be more widely applicable is to consider weak predictors that are unparallelized, *i.e.*single threaded, and derive algorithms which select weak predictors to run in parallel. The anytime algorithms we presented here can naively be parallelized by simply scheduling the sequences of weak predictors selected using some scheduling policy, but it is unclear what kind of guarantees can be derived here, and if some joint learning of scheduling policy and weak predictors to use in the ensemble could perform better.

One other setting to consider would be one more similar to the web service setting used as an illustrative example. In this setting all weak predictors can potentially be run in parallel, but the cost of evaluating a weak predictor may be dependent on what portion of the data is sent to it. In this setting the learning problem would be to find a policy for evaluating different weak predictors on a given example, where weak predictors can also be evaluated in parallel. For example, it may best to evaluate a single simple weak predictor on all examples, then, depending on the outcome, evaluate many expensive weak predictors in parallel. It may be that the same algorithms can be used for both of these parallel settings. Investigating anytime prediction algorithms for this setting would be interesting as well.

8.1.3 Branching Predictors

Many recent approaches to the budgeted prediction problem utilize a branching approach which can compute different weak predictors, or use different actions, on each example based on previous predictions. For example, policy based approaches [Busa-Fekete et al., 2012, Karayev et al., 2012, He et al., 2013] learn a policy which uses previous predictions to select which actions to take next, and hence can select to compute different predictors based on the outcome of early predictive actions. Similarly, Gao and Koller [2011] use an approach which conditions actions on previous predictions, and Xu et al. [2013b] give an extension of the Greedy Miser approach for learning a

tree of weak predictors instead of a sequence.

Many of these approaches observe a significant decrease in cost for achieving the same accuracy, because individual weak predictors can be targeted to different subsets of the input space. To compare with these approaches and explore possible gains from branching, it would be interesting to consider a branching version of our anytime approach, which learn a tree of weak predictors in a manner similar to Xu et al. [2013b]. To learn their tree based structure they utilize a global optimization which adjusts all nodes in the tree simultaneously and optimize final performance. It would be interesting to see if the greedy approach here could be adapted to select a tree of predictors, or if some kind of global optimization of decisions must be done to obtain efficient tree-based performance.

8.1.4 Understanding Generalization Properties

Throughout this document we have analyzed the predictive performance of our learned anytime predictors using the training performance as a metric, and the near-optimality guarantees given are statements about the near-optimality of cost-greedy algorithms with respect to the optimal training performance. However, in practice, we often see that these cost-greedy approaches cause an increase in overfitting, particularly in the domains where computation time is dominated by feature computation costs, by emphasizing the maximal re-use of already computed features.

A useful line of work would be to analyze the generalization properties of our anytime approach, and possibly improve the robustness to overfitting by modifying our algorithms. We have developed some methods for doing this in our practical applications, such as evaluating the cost-greedy metric on held-out validation data, but there is still much work to be done in understanding how to handle overfitting in general.

Bibliography

D. M. Bradley. *Learning in Modular Systems*. PhD thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2009.

G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla. Segmentation and recognition using structure from motion point clouds. In *ECCV*, 2008.

S. Brubaker, J. Wu, J. Sun, M. Mullin, and J. Rehg. On the design of cascades of boosted ensembles for face detection. *International Journal of Computer Vision*, pages 65–86, 2008.

R. Busa-Fekete, D. Benbouzid, and B. Kegl. Fast classification using sparse decision dags. In *Proceedings of the 28th International Conference on Machine Learning*, 2012.

B. B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *Proceedings of the Third International Conference on Web Search and Web Data Mining (WSDM)*, pages 411–420, 2010.

J. Chen and X. Huo. Theoretical Results on Sparse Representations of Multiple-Measurement Vectors. *IEEE Transactions on Signal Processing*, 54(12):4634–4643, Dec. 2006.

M. Chen, Z. Xu, K. Weinberger, O. Chapelle, and D. Kedem. Classifier cascade: Tradeoff between accuracy and feature evaluation cost. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, April 2012.

S. Chen, D. Donoho, and M. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43 (1):129–159, Jan. 2001.

A. Coates, H. Lee, and A. Y. Ng. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, 2011.

W. W. Cohen and V. Carvalho. Stacked sequential learning. In *IJCAI*, 2005.

S. Cotter, B. Rao, K. Engan, and K. Kreutz-Delgado. Sparse solutions to linear inverse problems with multiple measurement vectors. *Signal Processing, IEEE Transactions on*, 53(7):2477–2488, July 2005.

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292, March 2002.

A. Das and D. Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

H. Daume III, J. Langford, and D. Marcu. Search-based structured prediction. *MLJ*, 75(3), 2009.

R. de Nijs, S. Ramos, G. Roig, X. Boix, L. Van Gool, and K. Kuhnlenz. On-line semantic perception using uncertainty. In *IROS*, 2012.

N. Duffy and D. Helmbold. Potential boosters? In *Advances in Neural Information Processing Systems 12*, pages 258–264, Cambridge, MA, 2000. MIT Press.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.

C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. In *T-PAMI*, 2013.

P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *IJCV*, 59 (2), 2004.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting,. *Journal of Computer and System Sciences*, 55(1):119 – 139, 1997.

Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

J. H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38: 367–378, 1999.

J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2000.

T. Gao and D. Koller. Active classification based on value of classifier. In *Advances in Neural Information Processing Systems (NIPS 2011)*, 2011.

S. A. V. D. Geer and P. Buhlmann. On the conditions used to prove oracle results for the lasso. *Electronic Journal of Statistics*, 3:1360–1392, 2009.

S. Gould, J. Rodgers, D. Cohen, G. Elidan, and D. Koller. Multi-class segmentation with relative location prior. *IJCV*, 80(3), 2008.

S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semantically consistent regions. In *ICCV*, 2009.

A. Grubb and J. A. Bagnell. Boosted backpropagation learning for training deep modular networks. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

A. Grubb and J. A. Bagnell. Generalized boosting algorithms for convex optimization. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

A. Grubb and J. A. Bagnell. Speedboost: Anytime prediction with uniform near-optimality. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, April 2012.

T. Hastie and B. Efron. lars: Least angle regression, lasso and forward stagewise, 2013. URL `cran.r-project.org/web/packages/lars/`.

E. Hazan, A. Kalai, S. Kale, and A. Agarwal. Logarithmic regret algorithms for online convex optimization. In *Proceedings of the 19th Annual Conference on Learning Theory*, pages 499–513, 2006.

H. He, H. Daume III, and J. Eisner. Dynamic feature selection for dependency parsing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, Oct. 2002. ISSN 1046-8188.

J. Jiang, A. Teichert, H. Daume III, and J. Eisner. Learned prioritization for trading off accuracy and speed. In *NIPS*, 2012.

A. Juditsky and A. Nemirovski. Functional aggregation for nonparametric regression. *Annals of Statistics*, 28:681–712, 2000.

S. Karayev, T. Baumgartner, M. Fritz, and T. Darrell. Timely object recognition. In *NIPS*, 2012.

S. Khuller, A. Moss, and J. Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, Apr. 1999.

A. Krause and V. Cehver. Submodular dictionary selection for sparse representation. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.

A. Krause and D. Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2012.

A. Krause and C. Guestrin. A note on the budgeted maximization on submodular functions. Technical Report CMU-CALD-05-103, Carnegie Mellon University, 2005.

S. Kumar and M. Hebert. Discriminative random fields. *IJCV*, 68(2), 2006.

L. Ladicky. *Global Structured Models towards Scene Understanding*. PhD thesis, Oxford Brookes University, 2011.

L. Ladicky, P. Sturgess, K. Alahari, C. Russell, and P. H. Torr. What, where & how many? combining object detectors and crfs. In *ECCV*, 2010.

J. Langford. Vowpal wabbit, 2013. URL `http://hunch.net/~vw/`.

J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 420–429, 2007.

H. Lin and J. Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *HLT-NAACL*, pages 912–920. The Association for Computational Linguistics, 2010.

S. Mallat and Z. Zhang. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on*, 41(12):3397 –3415, dec 1993.

L. Mason, J. Baxter, P. L. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. In *Advances in Large Margin Classifiers*. MIT Press, 1999.

G. Mathews. On the partition of numbers. In *Proceedings of the London Mathematical Society*, volume 28, pages 486–490, 1897.

L. Meier, S. Van De Geer, and P. Bhlmann. The group lasso for logistic regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(1):53–71, 2008.

Microsoft. Microsoft learning to rank datasets, 2010. URL `http://research.microsoft.com/en-us/projects/mslr`.

A. Miller. *Subset Selection in Regression*. Chapman and Hall, second edition, 2002.

I. Mukherjee and R. E. Schapire. A theory of multiclass boosting. In *Advances in Neural Information Processing Systems 22*, Cambridge, MA, 2010. MIT Press.

D. Munoz, J. A. Bagnell, and M. Hebert. Stacked hierarchical labeling. In *European Conference on Computer Vision*, 2010.

B. Natarajan. Sparse approximation solutions to linear systems. *SIAM Journal on Computing*, 24: 227–234, 1995.

G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 14:265–294, 1978.

Y. C. Pati, R. Rezaiifar, Y. C. P. R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27 th Annual Asilomar Conference on Signals, Systems, and Computers*, pages 40–44, 1993.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

A. Rakotomamonjy. Surveying and comparing simultaneous sparse approximation (or group-lasso) algorithms. *Signal Processing*, 91:1505–1526, 2011.

N. Ratliff. *Learning to Search: Structured Prediction Techniques for Imitation Learning*. PhD thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2009.

N. Ratliff, D. Silver, and J. A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, 27(1):25–53, July 2009.

G. Rätsch, S. Mika, and M. K. Warmuth. On the convergence of leveraging. In *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

L. Reyzin. Boosting on a budget: Sampling for feature-efficient prediction. In *Proceedings of the 28th Internation Conference on Machine Learning (ICML)*, 2011.

S. Ross, D. Munoz, M. Hebert, and J. A. Bagnell. Learning message-passing inference machines for structured prediction. In *CVPR*, 2011.

S. Ross, J. Zhou, Y. Yue, D. Dey, and J. A. Bagnell. Learning policies for contextual submodular prediction. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

M. J. Saberian and N. Vasconcelos. Boosting classifier cascades. In *Proceedings of the 24th Annual Conference on Neural Information Processing Systems (NIPS)*, 2010.

R. E. Schapire. The boosting approach to machine learning: An overview. In *MSRI Workshop on Nonlinear Estimation and Classification*, 2002.

R. Socher, C. Lin, A. Y. Ng, and C. D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011.

J. Sochman and J. Matas. Waldboost: Learning for time constrained sequential detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 150–156, 2005.

B. Sofman, J. A. Bagnell, and A. Stentz. Anytime online novelty detection for vehicle safeguarding. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (ICRA)*, 2010.

M. Streeter and D. Golovin. An online algorithm for maximizing submodular functions. In *Proceedings of the 22nd Annual Conference on Neural Information Processing Systems (NIPS)*, 2008.

P. Sturgess, K. Alahari, L. Ladicky, and P. H. S. Torr. Combining appearance and structure from motion features for road scene understanding. In *BMVC*, 2009.

P. Sturgess, L. Ladicky, N. Crook, and P. H. S. Torr. Scalable cascade inference for semantic image segmentation. In *BMVC*, 2012.

I. Sutskever. A simpler unified analysis of budget perceptrons. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 985–992, New York, NY, USA, 2009. ACM.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of Royal Statistical Society*, 58:267–288, 1996.

J. Tighe and S. Lazebnik. Superparsing: Scalable nonparametric image parsing with superpixels. *IJCV*, 2013.

J. A. Tropp. Greed is good: Algorithmic results for sparse approximation. *IEEE Trans. Inform. Theory*, 50:2231–2242, 2004.

J. A. Tropp, A. C. Gilbert, and M. J. Strauss. Algorithms for simultaneous sparse approximation. part i: Greedy pursuit. *Signal Processing*, 86(3):572–588, 2006.

Z. Tu and X. Bai. Auto-context and its application to high-level vision tasks and 3d brain image segmentation. *T-PAMI*, 32(10), 2010.

K. Ueno, X. Xi, E. Keogh, and D. Lee. Anytime classification using the nearest neighbor algorithm with applications to stream mining. *Data Mining, IEEE International Conference on*, 0:623–632, 2006.

P. Viola and M. J. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2001.

P. A. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, 57(2), 2004.

D. Weiss and B. Taskar. Structured prediction cascades. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, 2010.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2), 1992.

Z. Xu, K. Weinberger, and O. Chapelle. The greedy miser: Learning under test-time budgets. In *Proceedings of the 28th International Conference on Machine Learning*, 2012.

Z. Xu, M. Kusner, G. Huang, and K. Q. Weinberger. Anytime representation learning. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1076–1084. JMLR Workshop and Conference Proceedings, 2013a.

Z. Xu, M. Kusner, K. Q. Weinberger, and M. Chen. Cost-sensitive tree of classifiers. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 131–141. JMLR Workshop and Conference Proceedings, 2013b.

Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. In *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2007. MIT Press.

S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

G. Zoutendijk. Nonlinear programming, computational methods. In J. Abadie, editor, *Integer and nonlinear programming*, pages 37–86. North-Holland: Amsterdam, 1970.